AN ABSTRACT OF THE DISSERTATION OF

Wenxuan Wu for the degree of Doctor of Philosophy in Computer Science presented on April 28, 2022.

Title: Deep Convolutional Network on Point Clouds for 3D Scene Understanding

Abstract approved: _____

Fuxin Li

As one of the most popular data types, the point cloud is widely used in various applications, including computer vision, computer graphics and robotics. The capability to directly measure 3D point clouds is invaluable in those applications as depth information could remove a lot of the segmentation ambiguities in 2D images. Unlike images which are represented in regular dense grids, 3D point clouds are irregular and unordered, hence applying convolution on them can be difficult. To address this problem, we extend the dynamic filter to a new convolution operation, named PointConv. PointConv can be applied on point clouds to build deep convolutional networks. We treat convolution kernels as nonlinear functions of the local coordinates of 3D points comprised of weight and density functions. With respect to a given point, the weight functions are learned with multi-layer perceptron networks, and density functions through kernel density estimation. The most important contribution of this work is a novel reformulation proposed for efficiently computing the weight functions, which allowed us to dramatically scale up the network and significantly improve its performance. The learned convolution kernel can be used to compute translation-invariant and permutation-invariant convolution on any point set in the 3D space.

The proposed PointConv have opened doors to new 3D-centric approaches to scene understanding. We show how we can adapt and apply PointConv to an important perception problem in robotics: 3D scene flow estimation. We propose a novel end-to-end deep scene flow model, called PointPWC-Net, that directly processes 3D point cloud

scenes with large motions in a coarse-to-fine fashion. Flow computed at the coarse level is upsampled and warped to a finer level, enabling the algorithm to accommodate for large motion without a prohibitive search space. We introduce novel cost volume, upsampling, and warping layers to efficiently handle 3D point cloud data. Unlike traditional cost volumes that require exhaustively computing all the cost values on a high-dimensional grid, our point-based formulation discretizes the cost volume onto input 3D points, and a PointConv operation efficiently computes convolutions on the cost volume.

Finally, inspired by the recent development of Transformer, We introduce PointConvFormer, a novel building block for point cloud based deep neural network architectures. PointConvFormer combines ideas from point convolution, where filter weights are only based on relative position, and Transformers where the attention computation takes the features into account. In our proposed new operation, feature difference between points in the neighborhood serves as an indicator to re-weight the convolutional weights. Hence, we preserved some of the translation-invariance of the convolution operation whereas taken attention into account to choose the relevant points for convolution. We also explore multi-head mechanisms as well. To validate the effectiveness of PointConvFormer, we experiment on both semantic segmentation and scene flow estimation tasks on point clouds with multiple datasets including ScanNet, SemanticKitti, FlyingThings3D and KITTI. Our results show that PointConvFormer substantially outperforms classic convolutions, regular transformers, and voxelized sparse convolution approaches with smaller, more computationally efficient networks.

Deep Convolutional Network on Point Clouds for 3D Scene Understanding

by
Wenxuan Wu

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented April 28, 2022
Commencement June 2022

Doctor of Philosophy dissertation of Wenxuan Wu presented on April 28, 2022.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Wenxuan Wu, Author

ACKNOWLEDGEMENTS

Throughout the writing of this thesis I have received a great deal of support and assistance. This thesis could not happen without all the help and support. First and foremost, I would first like to thank my PhD advisor Dr. Fuxin Li for the support, encouragement and mentorship. Fuxin has been an extraordinary advisor and an academic role model to me. I really appreciate Fuxin's openness and support to my projects and research topics. He always encourages me to think deeper and go beyond architecture engineering in research. I also appreciate the freedom Fuxin gives me, both in choosing research topics and in selecting a career path. Fuxin gives his advice but lets me to decide what I really want to do. I am often surprised by how knowledgeable Fuxin is and always get the advises I need for my projects.

I am also thankful for Prof. Sinisa Todorovic, Prof. Alan Fern, Prof. Lizhong Chen, Prof. Hao Su and Prof. Yelda Turkan being on my PhD committee. I really appreciate your responsiveness in all communications and your time and effort during any exams and meetings.

I would like to thank my great lab mates Zhongang, Lawrence, Neale, Robert, Xingyi, Ali, Amin, Hung, Jay, Jialin, Xianfang, Mazen, Michael, Tim, Vijay, Saeed, Xinyao, Zehuan, and Ziwen. They not only helped tremendously with feedback and advice, but they gracefully listened to the same presentations and practice talks so many times from me. I would extend a special thanks to Dr. Zhongang Qi, who gives many interesting suggestions and help to my first research work in Oregon State University. Besides, I also would like to give thanks to Jialin Yuan, who guided me through the initial onboarding process of my PhD and also gave me a lot of suggestions and life experiences.

I am grateful to the great colleagues, all my course instructors, and excellent faculty from the EECS department of Oregon State University. I have learned a lot from the questions we discussed. I thank all the staff members in the department for responding to all my requests and tracking my progress.

Besides, I want to thank Nuro Inc., Waymo LLC., Facebook Reality Lab and Apple Inc. for providing me memorable internship experiences during the summer term of my PhD.

Great research is often motivated by great needs. These internships motivated me to explore more on deep learning on 3D point clouds including 3D scene flow estimation and more interesting topics. I am very grateful to my mentors and co-workers during the internships: Wei Liu, Zhiyuan Wang, Zhuwen Li, Xiao Zhang, Jing Dong, and Yipu Zhao. I learned a lot from them including technical knowledge, communications and teamwork.

Finally, I would like to give sincere thanks to my parents, my sisters and all the family members. My family are always with me. Without their support and love, I wouldn't be able to achieve what I achieve now. Deeply thanks to my parents for always support me to choose what I love to do. I also thank my sisters for providing me career and life suggestions. Besides, I also would like to give thanks to my grandmother for taking care of me when I was little and giving me a kind and patient heart.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

LIST OF TABLES (Continued)

Table Page

LIST OF TABLES (Continued)

Table Page

LIST OF ALGORITHMS

Chapter 1: Introduction

1.1 Motivation

In recent robotics, autonomous driving and virtual/augmented reality applications, sensors that can directly obtain 3D data are increasingly ubiquitous. This includes indoor sensors such as laser scanners, time-of-flight sensors such as the Kinect, RealSense or Google Tango, structural light sensors such as those on the iPhoneX, as well as outdoor sensors such as LIDAR and MEMS sensors. The capability to directly measure 3D data is invaluable in those applications as depth information could remove a lot of the segmentation ambiguities from 2D imagery, and surface normals provide important cues of the scene geometry.

In 2D images, convolutional neural networks (CNNs) have fundamentally changed the landscape of computer vision by greatly improving results on almost every vision task. CNNs succeed by utilizing translation invariance, so that the same set of convolutional filters can be applied on all the locations in an image, reducing the number of parameters and improving generalization. We would hope such successes to be transferred to the analysis of 3D data. However, 3D data often come in the form of point clouds, which is a set of unordered 3D points, with or without additional features (e.g. RGB) on each point. Point clouds are unordered and do not conform to the regular lattice grids as in 2D images. It is difficult to apply conventional CNNs on such unordered input.

There are some work converting 3D point clouds into certain structured data format for the ease of processing. [124, 107] propose to project 3D point clouds or shapes into several 2D images, and then apply 2D convolutional networks for classification. Due to projection, certain important 3D information is lost and it is not always obvious which viewpoint to select for better performance. [160, 89, 107] represent another type of approach that voxelizes point clouds into volumetric grids by quantization and then apply 3D convolution networks. However, those methods suffers greatly from the memory consumption and computation efficiency. Most of the proposed methods just work on a

very coarse volume, such as 30 by 30 by 30 in resolution [107], which in turn cause large quantization errors and information loss.

Hence, we aim to seek operations that work as basic operations for 3D point clouds as the 2D convolution for images. This operation should be able to directly take 3D point clouds as input without information loss, and be able to stack multiple layers to build deep neural network for high representation capabilities. This fundamental operation works just like bricks to a building. With that, we are able to open a new world to the 3D deep learning. The first attempt of this work is PointConv, which is a Monte Carlo approximation of the continuous convolution on 3D point clouds. This operation is a convolution that can be directly applied on 3D point cloud input, which leads to similar representation capabilities as 2D convolution. To reduce the memory consumption of the operation and build deep neural network at scale, we further propose a memory efficient version of PointConv, which could be used to build super deep neural networks. As we know, deep neural networks, such as ResNet [44], are backbones for almost all the tasks in computer vision. With the efficient PointConv, we are able to build PointConv version of ResNet [44] and use it as basic network backbones to address different tasks in 3D computer vision.

As mentioned above, with the help of efficient PointConv, it is possible to explore more complicated problems in the 3D Computer Vision field. Estimating scene flow directly from 3D point clouds is an interesting and challenging task. As a fundamental low-level understanding of the world, scene flow can be used in various applications, such as motion segmentation, visual odometry, action recognition, and autonomous driving, etc. Traditionally, most scene flow estimation algorithms take 2D images as input. They first estimate the depth map and the optical flow from the input image pairs. Then, the scene flow is reconstructed from the depth map and the optical flow. The methods usually require multiple deep neural networks, multi-task training, and complicate post-processing, etc. Due to the ambiguous nature of 2D images, the methods usually require strong assumptions for better estimation results, such as piece-wise rigid, etc.

With the development of deep learning in 3D point clouds, it is possible to estimate scene flow directly from 3D point clouds in an end-to-end trainable fashion [78, 42]. Fundamentally, scene flow estimation from 3D point clouds is a dense matching problem

with inputs being time sequence point cloud frames. Most of the matching problems in computer vision require computing the cost volume [128, 134]. In images, the pixels are always aligned in grid shapes, which is regular and straightforward to process. The cost volume is computed simply by aggregating the correlation in patches [162]. On the other hand, 3D point clouds are more flexible and irregular [106]. It is unclear how to aggregate the correlation to construct a cost volume. To address this problem, we are the first to propose a novel cost volume layer that can directly be discretized onto the irregular 3D point clouds with the help of PointConv. The new cost volume takes the neighbourhood from both point clouds into account and utilizes the efficient PointConv to aggregate the cost values. As a result, the novel cost volume contains important matching information and is very efficient to compute. With the design of the novel cost volume, we unlock the probabilities of adopting many different network structures, such as PWC-Net [128], into the scene flow estimation from 3D point clouds. Furthermore, any matching problems, such as geometry matching and instance matching [138], that require computing the cost volume could potentially utilize our novel cost volume to further boost the performance. Besides the cost volume layer, we also propose methods to build robust feature pyramids on 3D point clouds, introduce the warping operation to reduce the searching area, and a simple but effective method to upsample the scene flow from a coarse level to a finer one. With all the previously mentioned layers, we introduce the point cloud version of PWC-Net [128], which we named it PointPWC-Net [159].

One of the key components when conducting convolution on 3D point clouds is the construction of the local neighbourhood with high correlation points. Due to the irregular structure of point clouds, people heavily rely on the K-nearest neighbour(KNN) algorithm to compute the neighbourhood around a convolutional center. However, the shape of the neighbourhood varies in different parts of the point cloud. The neighbourhood point nearby does not necessarily have high relevance to the neighbourhood center in feature space, especially in object boundaries, where the feature could be very different. With less-relevant points in the convolutional neighbourhood, the output feature could be "blurred" or "corrupted", which leads to fewer representation capabilities or strong ambiguity. To address this issue, we introduce feature-based weights to PointConv, which serves as a re-weighted filter to filter out the less-relevant features in the neighbourhood. With that in mind, we find that the re-weighted filter shares some similarities with the

recent popular vision transformer [29], whose aggregation weights are also computed from feature space. As a result, our proposed PointConvFormer combines the properties of PointConv, whose weights are computed from low dimensional spatial space, and the properties of transformers, whose weights are computed from high dimensional feature space. The PointConvFormer addresses the puzzle of how to define a "good" neighbourhood in point cloud processing for better representation and generalization.

1.2 Overview

In our work *PointConv*(Chapter 2), we propose a novel convolution operation which directly works on point cloud data. PointConv takes the point cloud along with its features as input and output encoded features for deeper layers. Different from previous methods, we take point clouds as a specific sampling or discretization from continuous surfaces with respect to different sensors. Hence, we can adopt the continuous convolution onto point clouds by discretize the continuous convolutional weights. In PointConv, the convolutional weights are represented as continuous functions. Because point cloud is a flexible data format, the local neighbourhood in the point cloud varies in different positions. For each local neighbourhood of the point cloud, the continuous convolutional weights is discretized with respect to the local neighbourhood. With the weights and the input features, we can conduct the convolution on the point cloud. To further improve the efficiency of PointConv, we propose a novel reformulation to implement PointConv by reducing the original PointConv formulation to two standard operations–matrix multiplication and 2D convolution. This new computation trick greatly reduces the memory consumption and is fully equivalent to the original PointConv. With the efficient PointConv, we are able to build deep convolutional neural networks for various computer vision tasks, such as classification, semantic segmentation, etc. Besides working on 3D point clouds, our PointConv can also be applied on 2D images by treating pixels of images as 2D points of point clouds. We conduct experiments on the popular Cifar10 dataset [64] and achieve on par results comparing with standard 2D convolution using similar network structures.

To further evaluate the performance of PointConv and study its properties, we apply PointConv to the problem of scene flow estimation from 3D point clouds. Scene flow is

the 3D displacement vector field between two input point clouds [165], which represents the motion between two 3D scenes. Scene flow is a low-level computer vision task, which could be used to further improve the performance of high-level vision task, such as object detection, action recognition and tracking, etc. Previously, most methods estimate scene flow from RGB or RGBD data. With the recent improvement of the 3D sensors, directly estimating scene flow from 3D point clouds has drawn more and more attention.

However, due to different data structures between 2D images and 3D point clouds, the method to estimate scene flow directly from 3D point clouds is still a open problem. In our work *PointPWC-Net*(Chapter 3), we explore the coarse-to-fine framework [8, 12, 125, 127] for 3D point clouds. The coarse-to-fine framework in deep neural network is first introduced in PWC-Net [128], which computes the coarse estimation first and gradually refines the results to finer level. As a result, the coarse-to-fine framework allows us to accommodate large motion to a coarse level without a prohibitive search space, which is both memory and computation efficient. To take advantages of the efficiency of the coarse-to-fine framework, we propose PointPWC-Net in Chapter 3.

One of the key components in scene flow estimation is the cost volume computation. The cost volume contains the matching information between neighbouring pixel pairs from consecutive frames. Unlike cost volumes from image pairs, the cost volumes from 3D point cloud pairs are still under explored due to the flexibility of the point cloud data format. In Chapter 3, we propose a novel cost volume for 3D point clouds along with a learnable cost representation between two neighbouring points from consecutive point cloud frames. The new cost volume is computed with two PointConv operations, which is both memory and time efficient for flow estimation. In order to build the PointPWC-Net, we further introduce the upsampling layers, warping layers to interpolate and warp the flow within pyramid levels. With all the novel layers, we are able to build a PointPWC-Net for accurate estimation of scene flow directly from 3D point clouds.

As in optical flow, it is difficult to obtain accurate flow ground truth labels for point cloud data. To train the PointPWC-Net, we propose a novel self-supervised loss to train the model without any ground truth flow. The new self-supervised loss contains the Chamfer distance, Smoothness constraint, and Laplacian regularization. With the loss terms, we are able to train the model to achieve state-of-the-art performance without

any supervision.

Recently, researchers have introduced visual transformers to 3D point clouds processing, inspired by the success in NLP and image analysis [177, 102]. The self-attention mechanism has also been adapted for point clouds processing [177]. However, so far, transformers have not shown to significantly outperform convolutional approaches on point clouds. Both convolution and attention aim to conduct feature aggregation in neighborhoods with high feature correlations. The advantage of convolution is that its translation-invariance usually offers good generalization power. However, attention weights can help locate points that are more correlated with each other.

In Capter 4, we propose *PointConvFormer*, an operation that uses attention weights to modulate a convolution operation, essentially selecting relevant points to perform convolution, with the hope to get the best of both worlds. We also experiment with the multi-head mechanism commonly used in transformers. To demenstrate the effectiveness of the PointConvFormer, we evaluate PointConvFormer on two point cloud tasks, semantic segmentation and scene flow estimation. For semantic segmentation, experiment results on the indoor ScanNet [23] and the outdoor SemanticKitti [7] demonstrate superior performances over classic convolution and transformers with a more compact network. We also apply PointConvFormer as the backbone of PointPWC-Net [159] for scene flow estimation, and observe significant improvements on FlyingThings3D [90] and KITTI scene flow 2015 [96] datasets.

1.3 Contributions and Thesis Outline

Figure 1.1 illustrates the contributions of the thesis. We summary the key contributions as follows:

• We propose a novel convolutional operation for point clouds processing–PointConv. PointConv is memory efficient and can be used to further build deep neural network for various computer vision tasks. Extensive experiments are conducted to demonstrate the effective of the novel PointConv.

• We extend the coarse-to-fine framework to the scene flow estimation directly from 3D point clouds, called PointPWC-Net. PointPWC-Net consists feature pyramid, learnable

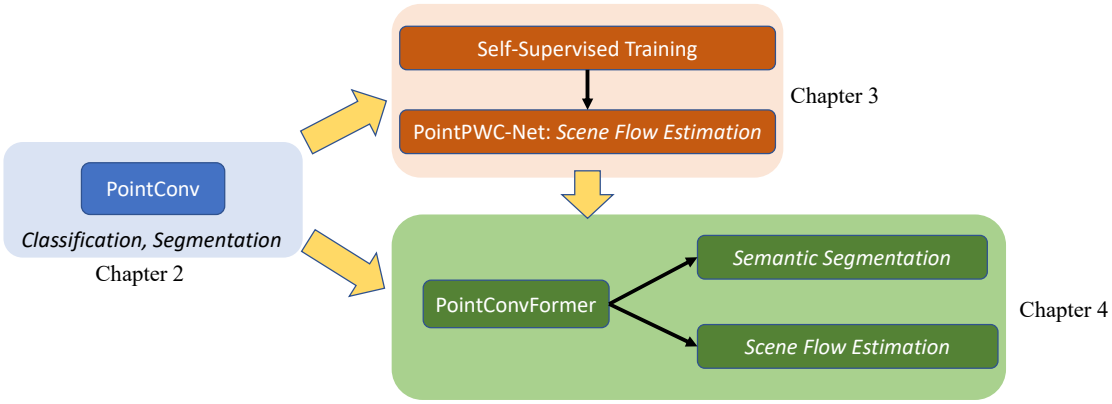


Figure 1.1: Thesis contributions. The main contributions of the thesis include the convolutional operation–PointConv, the scene flow estimation network–PointPWC-Net, and the improved convolutional operation–PointConvFormer.

cost volume layers, upsampling layers, warping layers, and scene flow predictors. Besides, to train PointPWC-Net without any ground truth annotations, we propose a new self-supervised loss. Experiments show that our novel network is able to outperforms the state-of-the-art methods with a large margin.

• To further improve the performance of dense prediction models, we propose an improved convolution–PointConvFormer, whose weights take the relative positions and the correlation of the guidance features into consideration. The PointConvFormer is applied to 3D semantic segmentation and scene flow estimation from 3D point clouds.

The outline of the thesis are organized as follows:

Chapter 2 introduces a novel convolutional operation that can directly processes 3D point clouds. The convolutional operation is named PointConv. Besides, we also extend the PointConv to PointDeConv, which is used for feature propagation. To improve the efficiency of PointConv, we introduce the PointConv to an efficient version, which can be used to build deep network at scale. The content of this chapter is primarily based on [158].

Chapter 3 applies the PointConv into a 3D computer vision task, Scene flow estimation from 3D point clouds. A novel network structure, called PointPWC-Net, is introduced

for efficient scene flow estimation. PointPWC-Net is able to outperform the state-of-the-art methods by a large margin with the multi-scale supervised loss. However, due to the fact that the dense annotation of scene flow for 3D point clouds is hard, we further propose a novel self-supervised loss to train the network without any human annotation. Experiments on FlyingThings3D [90] and KITTI scene flow [95] demonstrate the effectiveness and generalization ability of the novel network.

Chapter 4 proposes a new convolutional operation, PointConvFormer. Unlike standard convolution, whose weights are functions of the relative positions of a local neighbourhood, the weights of PointConvFormer are represented as functions of the relative positions and the guidance features, which combines the properties of traditional convolution and the popular transformer. We conduct experiments on the 3D semantic segmentation tasks using PointConvFormer. Besides, we adapt PointConvFormer to the PointPWC-Net in Chapter 3 as GPointPWC-Net and achieve better performance on scene flow estimation.

Finally, we summarize the thesis and provide insights for possible future research directions and applications in **Chapter 5**.

Chapter 2: Deep Convolutional Networks on 3D Point Clouds

2.1 Introduction

In this chapter, we introduce a new approach to conduct convolution on 3D point clouds with non-uniform sampling. We note that the convolution operation can be viewed as a discrete approximation of a continuous convolution operator. In 3D space, we can treat the weights of this convolution operator to be a (Lipschitz) continuous function of the local 3D point coordinates with respect to a reference 3D point. The continuous function can be approximated by a multi-layer perceptron(MLP), as done in [118] and [59]. But these algorithms did not take non-uniform sampling into account. We propose to use an inverse density scale to re-weight the continuous function learned by MLP, which corresponds to the Monte Carlo approximation of the continuous convolution. We call such an operation **PointConv**. PointConv involves taking the positions of point clouds as input and learning an MLP to approximate a weight function, as well as applying a inverse density scale on the learned weights to compensate the non-uniform sampling.

The naive implementation of PointConv is memory inefficient when the channel size of the output features is very large and hence hard to train and scale up to large networks. In order to reduce the memory consumption of PointConv, we introduce an approach which is able to greatly increase the memory efficiency using a reformulation that changes the summation order. The new structure is capable of building multi-layer deep convolutional networks on 3D point clouds that have similar capabilities as 2D CNN on raster images. We can achieve the same translation-invariance as in 2D convolutional networks, and the invariance to permutations on the ordering of points in a point cloud.

In segmentation tasks, the ability to transfer information gradually from coarse layers to finer layer is important. Hence, a deconvolution operation [101] that can fully leverage the feature from a coarse layer to a finer layer is vital for the performance. Most state-of-the-art algorithms [106, 108] are unable to perform deconvolution, which restricts

their performance on segmentation tasks. Since our PointConv is a full approximation of convolution, it is natural to extend PointConv to a PointDeconv, which can fully untilize the information in coarse layers and propagate to finer layers. By using PointConv and PointDeconv, we can achieve improved performance on semantic segmentation tasks.

The contributions of our work are:

• We propose PointConv, a density re-weighted convolution, which is able to fully approximate the 3D continuous convolution on any set of 3D points.

• We design a memory efficient approach to implement PointConv using a change of summation order technique, most importantly, allowing it to scale up to modern CNN levels.

• We extend our PointConv to a deconvolution version(PointDeconv) to achieve better segmentation results.

Experiments show that our deep network built on PointConv is highly competitive against other point cloud deep networks and achieve state-of-the-art results in part segmentation [14] and indoor semantic segmentation benchmarks [23]. In order to demonstrate that our PointConv is indeed a true convolution operation, we also evaluate PointConv on CIFAR-10 by converting all pixels in a 2D image into a point cloud with 2D coordinates along with RGB features on each point. Experiments on CIFAR-10 show that the classification accuracy of our PointConv is comparable with a image CNN of a similar structure, far outperforming previous best results achieved by point cloud networks. As a basic approach to CNN on 3D data, we believe there could be many potential applications of PointConv.

2.2 Related Work

Most work on 3D CNN networks convert 3D point clouds to 2D images or 3D volumetric grids. [124, 107] proposed to project 3D point clouds or shapes into several 2D images, and then apply 2D convolutional networks for classification. Although these approaches have achieved dominating performances on shape classification and retrieval tasks, it is

nontrivial to extend them to high-resolution scene segmentation tasks [23]. [160, 89, 107] represent another type of approach that voxelizes point clouds into volumetric grids by quantization and then apply 3D convolution networks. This type of approach is constrained by its 3D volumetric resolution and the computational cost of 3D convolutions. [114] improves the resolution significantly by using a set of unbalanced octrees where each leaf node stores a pooled feature representation. Kd-networks[62] computes the representations in a feed-forward bottom-up fashion on a Kd-tree with certain size. In a Kd-network, the input number of points in the point cloud needs to be the same during training and testing, which does not hold for many tasks. SSCN [40] utilizes the convolution based on a volumetric grid with novel speed/memory improvements by considering CNN outputs only on input points. However, if the point cloud is sampled sparsely, especially when the sampling rate is uneven, for the sparsely sampled regions on may not be able to find any neighbor within the volumetric convolutional filter, which could cause significant issues.

Some latest work [112, 106, 108, 123, 133, 52, 41, 144] directly take raw point clouds as input without converting them to other formats. [106, 112] proposed to use shared multi-layer perceptrons and max pooling layers to obtain features of point clouds. Because the max pooling layers are applied across all the points in point cloud, it is difficult to capture local features. PointNet++ [108] improved the network in PointNet [106] by adding a hierarchical structure. The hierarchical structure is similar to the one used in image CNNs, which extracts features starting from small local regions and gradually extending to larger regions. The key structure used in both PointNet [106] and PointNet++ [108] to aggregate features from different points is max-pooling. However, max-pooling layers keep only the strongest activation on features across a local or global region, which may lose some useful detailed information for segmentation tasks. [123] presents a method that projects the input features of the point clouds onto a high-dimensional lattice, and then apply bilateral convolution on the high-dimensional lattice to aggregate features, which called "SPLATNet". The SPLATNet [123] is able to give comparable results as PointNet++ [108]. The tangent convolution [133] projects local surface geometry on a tangent plane around every point, which gives a set of planar-convolutionable tangent images. The pointwise convolution [52] queries nearest neighbors on the fly and bins the points into kernel cells, then applies kernel weights on the binned cells to convolve

on point clouds. Flex-convolution [41] introduced a generalization of the conventional convolution layer along with an efficient GPU implementation, which can applied to point clouds with millions of points. FeaStNet [144] proposes to generalize conventional convolution layer to 3D point clouds by adding a soft-assignment matrix. PointCNN [76] is to learn a $\chi-$transformation from the input points and then use it to simultaneously weight and permute the input features associated with the points. Comparing to our approach, PointCNN is unable to achieve permutation-invariance, which is desired for point clouds.

The work [118, 59, 154, 45, 151] and [163] propose to learn continuous filters to perform convolution. [59] proposed that the weight filter in 2d convolution can be treated as a continuous function, which can be approximated by MLPs. [118] firstly introduced the idea into 3d graph structure. [151] extended the method in [118] to segmentation tasks and proposed an efficient version, but their efficient version can only approximate depth-wise convolution instead of real convolution. Dynamic graph CNN [154] proposed a method that can dynamically updating the graph. [163] presents a special family of filters to approximate the weight function instead of using MLPs. [45] proposed a Monta Carlo approximation of 3D convolution by taking density into account. Our work differ from those in 3 aspects. Most importantly, our efficient version of a real convolution was never proposed in prior work. Also, we utilize density differently than [45], and we propose a deconvolution operator based on PointConv to perform semantic segmentation.

2.3 PointConv

We propose a convolution operation which extends traditional image convolution into the point cloud called **PointConv**. PointConv is an extension to the Monte Carlo approximation of the 3D continuous convolution operator. For each convolutional filter, it uses MLP to approximate a weight function, then applies a density scale to re-weight the learned weight functions. Sec. 2.3.1 introduces the structure of the PointConv layer. Sec. 2.3.3 introduces PointDeconv, using PointConv layers to deconvolve features.

2.3.1 Convolution on 3D Point Clouds

Formally, convolution is defined as in Eq.(2.1) for functions $f(\mathbf{x})$ and $g(\mathbf{x})$ of a d-dimensional vector $\mathbf{x}$.

$$(f * g)(\mathbf{x}) = \iint_{\boldsymbol{\tau} \in \mathbb{R}^d} f(\boldsymbol{\tau}) g(\mathbf{x} + \boldsymbol{\tau}) d\boldsymbol{\tau} \tag{2.1}$$

Images can be interpreted as 2D discrete functions, which are usually represented as grid-shaped matrices. In CNN, each filter is restricted to a small local region, such as $3 \times 3, 5 \times 5$, etc. Within each local region, the relative positions between different pixels are always fixed, as shown in Figure 2.1(a). And the filter can be easily discretized to a summation with a real-valued weight for each location within the local region.

A point cloud is represented as a set of 3D points $\{p_i | i = 1, ..., n\}$, where each point contains a position vector (x, y, z) and its features such as color, surface normal, etc. Different from images, point clouds have more flexible shapes. The coordinates $p = (x, y, z) \in \mathbb{R}^3$ of a point in a point cloud are not located on a fixed grid but can take an arbitrary continuous value. Thus, the relative positions of different points are diverse in each local region. Conventional discretized convolution filters on raster images cannot be applied directly on the point cloud. Fig. 2.1 shows the difference between a local region in a image and a point cloud.

To make convolution compatible with point sets, we propose a permutation-invariant convolution operation, called **PointConv**. Our idea is to first go back to the continuous version of 3D convolution as:

$$\begin{aligned} &Conv(W, F)_{xyz} = \\ &\iiint_{(\delta_x, \delta_y, \delta_z) \in G} W(\delta_x, \delta_y, \delta_z) F(x + \delta_x, y + \delta_y, z + \delta_z) d\delta_x \delta_y \delta_z \end{aligned} \tag{2.2}$$

where $F(x + \delta_x, y + \delta_y, z + \delta_z)$ is the feature of a point in the local region G centered around point $p = (x, y, z)$. A point cloud can be viewed as a non-uniform sample from the continuous $\mathbb{R}^3$ space. In each local region, $(\delta_x, \delta_y, \delta_z)$ could be any possible position

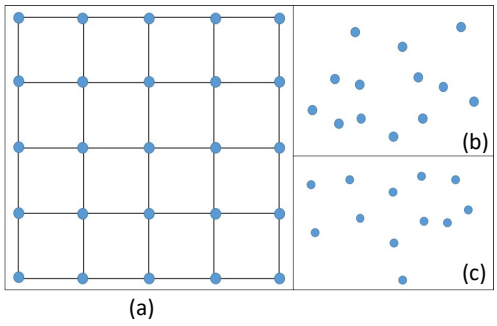


Figure 2.1: Image grid vs. point cloud. (a) shows a 5×5 local region in a image, where the distance between points can only attain very few discrete values; (b) and (c) show that in different local regions within a point cloud, the order and the relative positions can be very different.

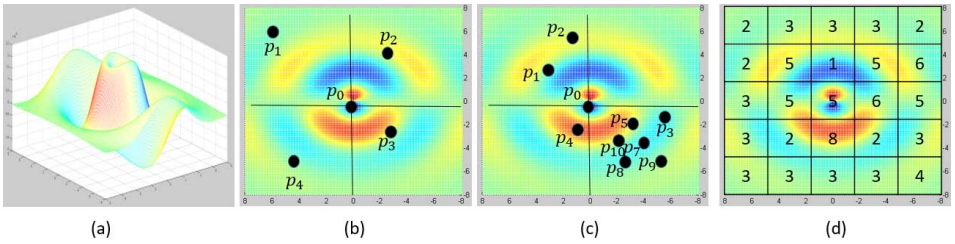


Figure 2.2: 2D weight function for PointConv. (a) is a learned continuous weight function; (b) and (c) are different local regions in a 2d point cloud. Given 2d points, we can obtain the weights at particular locations. The same applies to 3D points. The regular discrete 2D convolution can be viewed as a discretization of the continuous convolution weight function, as in (d).

in the local region. We define *PointConv* as the following:

$$PointConv(S, W, F)_{xyz} = \sum_{(\delta_x, \delta_y, \delta_z) \in G} S(\delta_x, \delta_y, \delta_z) W(\delta_x, \delta_y, \delta_z) F(x + \delta_x, y + \delta_y, z + \delta_z) \tag{2.3}$$

where $S(\delta_x, \delta_y, \delta_z)$ is the inverse density at point $(\delta_x, \delta_y, \delta_z)$. $S(\delta_x, \delta_y, \delta_z)$ is required because the point cloud can be sampled very non-uniformly[1]. Intuitively, the number

[1]To see this, note the Monte Carlo estimate with a biased sample: $\int f(x)dx = \int \frac{f(x)}{p(x)} p(x)dx \approx \sum_i \frac{f(x_i)}{p(x_i)}$, for $x_i \sim p(x)$. Point clouds are often biased samples because many sensors have difficulties

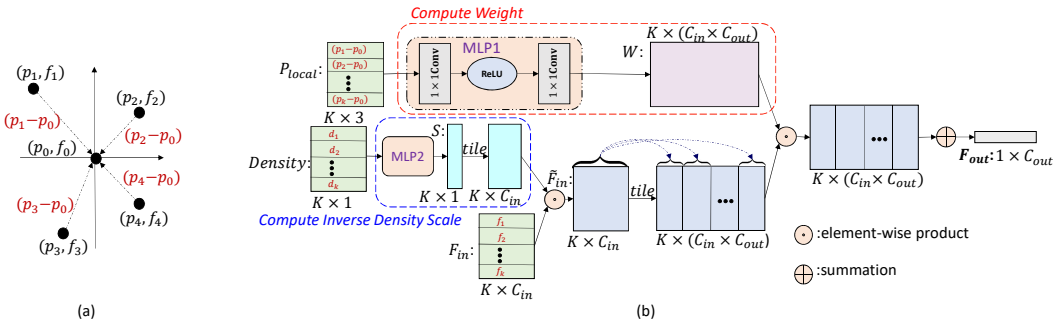


Figure 2.3: The structure of PointConv. (a) shows a local region with the coordinates of points transformed from global into local coordinates, p is the coordinates of points, and f is the corresponding feature; (b) shows the process of conducting PointConv on one local region centered around one point (p_0, f_0). The input features come form the K nearest neighbors centered at (p_0, f_0), and the output feature is F_{out} at p_0.

of points in the local region varies across the whole point cloud, as in Figure 2.2 (b) and (c). Besides, in Figure 2.2 (c), points $p_3, p_5, p_6, p_7, p_8, p_9, p_{10}$ are very close to one another, hence the contribution of each should be smaller.

Our main idea is to approximate the weight function $W(\delta_x, \delta_y, \delta_z)$ by multi-layer perceptrons from the 3D coordinates $(\delta_x, \delta_y, \delta_z)$ and and the inverse density $S(\delta_x, \delta_y, \delta_z)$ by a kernelized density estimation [139] followed by a nonlinear transform implemented with MLP. Because the weight function highly depends on the distribution of input point cloud, we call the entire convolution operation *PointConv*. [59, 118] considered the approximation of the weight function but did not consider the approximation of the density scale, hence is not a full approximation of the continuous convolution operator. Our nonlinear transform on the density is also different from [45].

The weights of the MLP in PointConv are shared across all the points in order to maintain the permutation invariance. To compute the inverse density scale $S(\delta_x, \delta_y, \delta_z)$, we first estimate the density of each point in a point cloud offline using kernel density estimation (KDE), then feed the density into a MLP for a 1D nonlinear transform. The reason to use a nonlinear transform is for the network to decide adaptively whether to use the density estimates.

measuring points near plane boundaries, hence needing this reweighting

Figure 2.3 shows the PointConv operation on a K-point local region. Let C_{in}, C_{out} be the number of channels for the input feature and output feature, k, c_{in}, c_{out} are the indices for k-th neighbor, c_{in}-th channel for input feature and c_{out}-th channel for output feature. The inputs are the 3D local positions of the points $P_{local} \in \mathbb{R}^{K\times 3}$, which can be computed by subtracting the coordinate of the centroid of the local region and the feature $F_{in} \in \mathbb{R}^{K\times C_{in}}$ of the local region. We use 1×1 convolution to implement the MLP. The output of the weight function is $W \in \mathbb{R}^{K\times(C_{in}\times C_{out})}$. So, $\mathbf{W}(k, c_{in}) \in \mathbb{R}^{C_{out}}$ is a vector. The density scale is $S \in \mathbb{R}^{K}$. After convolution, the feature F_{in} from a local region with K neighbour points are encoded into the output feature $\mathbf{F}_{out} \in \mathbb{R}^{C_{out}}$, as shown in Eq.(2.4).

$$\mathbf{F}_{out} = \sum_{k=1}^{K} \sum_{c_{in}=1}^{C_{in}} S(k)\mathbf{W}(k, c_{in})F_{in}(k, c_{in}) \tag{2.4}$$

PointConv learns a network to approximate the continuous weights for convolution. For each input point, we can compute the weights from the MLPs using its relative coordinates. Figure 2.2(a) shows an example continuous weight function for convolution. With a point cloud input as a discretization of the continuous input, a discrete convolution can be computed by Fig. 2.2(b) to extract the local features, which would work (with potentially different approximation accuracy) for different point cloud samples (Figure 2.2(b-d)), including a regular grid (Figure 2.2(d)). Note that in a raster image, the relative positions in local region are fixed. Then PointConv (which takes only relative positions as input for the weight functions) would output the same weight and density across the whole image, where it reduces to the conventional discretized convolution.

In order to aggregate the features in the entire point set, we use a hierarchical structure that is able to combine detailed small region features into abstract features that cover a larger spatial extent. The hierarchical structure we use is composed by several *feature encoding modules*, which is similar to the one used in PointNet++ [108]. Each module is roughly equivalent to one layer in a convolutional CNN. The key layers in each *feature encoding module* are the sampling layer, the grouping layer and the PointConv. More details can be found in the supplementary material.

The drawback of this approach is that each filter needs to be approximated by a network, hence is very inefficient. In Sec.2.4, we propose an efficient approach to implement

PointConv.

2.3.2 Hierarchical Structure

In order to aggregate the features in the entire point set, we use a hierarchical structure that is able to combine detailed small region features into abstract features that cover a larger spatial extent.

The hierarchical structure we use is composed by several *feature encoding modules*, which is similar to the one used in PointNet++ [108]. Each module is roughly equivalent to one layer in a convolutional CNN. We depict one feature encoding module in Alg.1.

Algorithm 1 Feature Encoding Module

Input : (P, F_{in}, N', K) : F_{in} is the features of the input point cloud P; N' is the number of points to keep after subsampling; K is the number of points in each local region.

Output: $(P_{centroid}, F_{out})$: $P_{centroid}$ is a subsample of the input point cloud P, which can be used as the input point clouds for next module; F_{out} is the corresponding feature.

1 Compute density at each point using KDE with respect to each point;
2 Feed density into DensityNet to get inverse density scale S;
3 Sample N' points out of input P, the N' points are the centriod points $P_{centroid}$ for each local region;
4 Find K nearest neighbor points P_{Knn} around each centroid point $P_{centroid}$. We denote the feature on the K nearest neighbors as F_{Knn} and the inverse density scale as S_{Knn};
5 Transform the global coordinates into local coordinates by subtracting each point with centroid point: $P_{localized} = P_{Knn} - P_{centroid}$;
6 Apply PointConv : $F_{out} \leftarrow PointConv(P_{localized}, F_{Knn}, S_{Knn})$

For simplicity, we consider the input point cloud with N points in the 3D space with Euclidean coordinates. The first step is to compute density of each input point using KDE. Then the density is fed into a MLP to get the inverse density scale S. The input point cloud is subsampled to get a point cloud with N' points. The subsampling method can be farthest point sampling [108], or inverse density sampling [41]. When $N' = N$, the output point cloud has the same size with the input point cloud, which means the module is a convolution with stride size = 1; when $N' < N$, the number of points in the output point cloud is smaller than the input point cloud, which means

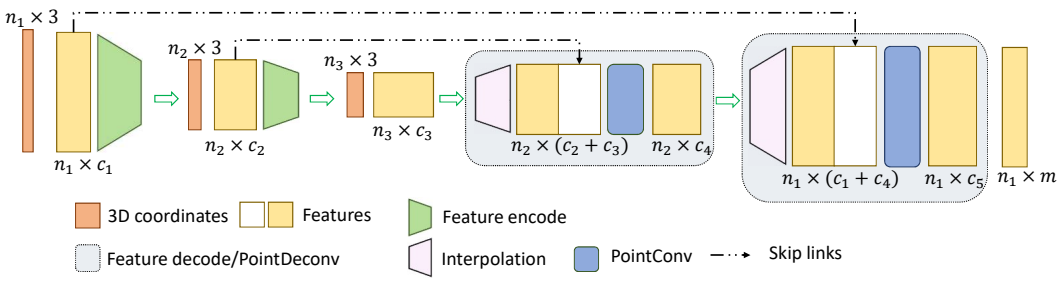


Figure 2.4: Feature encoding and propagation. This figure shows how the features are encoded and propagated in the network for a m classes segmentation task. n is the number of points in each layer, c is the channel size for the features. Best viewed in color.

the feature encoding module approximates a convolution with stride size > 1. The N' points serve as the centroid $P_{centroid}$ of each local region and are used to find their K nearest neighbor points P_{Knn}. The input features with shape $N \times C_{in}$ and the inverse density scale with shape $N \times 1$ are then grouped into local features F_{Knn} with shape $N' \times K \times C_{in}$ and inverse density scale S_{Knn} with shape $N' \times K \times 1$ according to their K nearest neighbor points. In order to apply the local weight function in PointConv layer, we convert the global coordinates into local coordinates by subtracting the centroid, $P_{localized} = P_{Knn} - P_{centroid}$. Then, the localized grouped points $P_{localized}$ are fed into a PointConv layer. After PointConv, the new feature F_{out} with shape $N' \times C_{out}$ together with the N' point cloud can be fed into the next *feature encoding module*.

2.3.3 Feature Propagation Using Deconvolution

For the segmentation task, we need point-wise prediction. In order to obtain features for all the input points, an approach to propagate features from a subsampled point cloud to a denser one is needed. PointNet++ [108] proposes to use distance-based interpolation to propagate features, which is reasonable due to local correlations inside a local region. However, this does not take full advantage of the deconvolution operation that captures local correlations of propagated information from the coarse level. We propose to add a PointDeconv layer based on the PointConv, as a deconvolution operation to address this issue.

As shown in Fig. 2.4, PointDeconv is composed of two parts: interpolation and PointConv. Firstly, we employ an interpolation to propagate coarse features from previous layer. Following [108], the interpolation is conducted by linearly interpolating features from the 3 nearest points. Then, the interpolated features are concatenated with features from the convolutional layers with the same resolution using skip links. After concatenation, we apply PointConv on the concatenated features to obtain the final deconvolution output, similar to the image deconvolution layer [101]. We apply this process until the features of all the input points have been propagated back to the original resolution.

2.4 Efficient PointConv

The naive implementation of the PointConv is memory consuming and inefficient. Different from [118], we propose a novel reformulation to implement PointConv by reducing it to two standard operations: matrix multiplication and 2d convolution. This novel trick not only takes advantage of the parallel computing of GPU, but also can be easily implemented using main-stream deep learning frameworks. Because the inverse density scale does not have such memory issues, the following discussion mainly focuses on the weight function.

Specifically, let B be the mini-batch size in the training stage, N be the number of points in a point cloud, K be the number of points in each local region, C_{in} be the number of input channels, and C_{out} be the number of output channels. For a point cloud, each local region shares the same weight functions which can be learned using MLP. However, weights computed from the weight functions at different points are different. The size of the weights filters generated by the MLP is $B \times N \times K \times (C_{in} \times C_{out})$. Suppose $B = 32$, $N = 512$, $K = 32$, $C_{in} = 64$, $C_{out} = 64$, and the filters are stored with single point precision. Then, the memory size for the filters is $8GB$ for only one layer. The network would be hard to train with such high memory consumption. [118] used very small network with few filters which significantly degraded its performance. To resolve this problem, we propose a memory efficient version of PointConv based on the following lemma:

Lemma 1. The PointConv is equivalent to the following formula: $\mathbf{F}_{out} = Conv_{1\times1}(\mathbf{H}, (\mathbf{S}\cdot$

$\mathbf{F}_{in})^T \otimes \mathbf{M}$) where $\mathbf{M} \in \mathbb{R}^{K \times C_{mid}}$ is the input to the last layer in the MLP for computing the weight function, and $\mathbf{H} \in \mathbb{R}^{C_{mid} \times (C_{in} \times C_{out})}$ is the weights of the last layer in the same MLP, $Conv_{1\times1}$ is 1×1 convolution.

Proof: Generally, the last layer of the MLP is a linear layer. In one local region, let $\widetilde{\mathbf{F}}_{\mathbf{in}} = \mathbf{S} \cdot \mathbf{F}_{\mathbf{in}} \in \mathbb{R}^{K \times C_{in}}$ and rewrite the MLP as a 1×1 convolution so that the output of the weight function is $\mathbf{W} = Conv_{1\times1}(H, M) \in \mathbb{R}^{K \times (C_{in} \times C_{out})}$. Let k is the index of the points in a local region, and c_{in}, c_{mid}, c_{out} are the indices of the input, middle layer and the filter output, respectively. Then $\mathbf{W}(k, c_{in}) \in \mathbb{R}^{C_{out}}$ is a vector from $\mathbf{W}$. And the $\mathbf{H}(c_{mid}, c_{in}) \in \mathbb{R}^{C_{out}}$ is a vector from $\mathbf{H}$. According to Eq.(2.4), the PointConv can be expressed in Eq.(2.5).

$$\mathbf{F}_{out} = \sum_{k=0}^{K-1} \sum_{c_{in}=0}^{C_{in}-1} (\mathbf{W}(k, c_{in}) \widetilde{\mathbf{F}}_{\mathbf{in}}(k, c_{in})) \tag{2.5}$$

Let's explore Eq.(2.5) in a more detailed manner. The output of the weight function can be expressed as:

$$\mathbf{W}(k, c_{in}) = \sum_{c_{mid}=0}^{C_{mid}-1} (\mathbf{M}(k, c_{mid}) \mathbf{H}(c_{mid}, c_{in})) \tag{2.6}$$

Substituting Eq.(2.6) into Eq.(2.5).

$$\begin{aligned} \mathbf{F}_{out} &= \sum_{k=0}^{K-1} \sum_{c_{in}=0}^{C_{in}-1} (\widetilde{\mathbf{F}}_{\mathbf{in}}(k, c_{in}) \sum_{c_{mid}=0}^{C_{mid}-1} (\mathbf{M}(k, c_{mid}) \mathbf{H}(c_{mid}, c_{in}))) \\ &= \sum_{c_{in}=0}^{C_{in}-1} \sum_{c_{mid}=0}^{C_{mid}-1} (\mathbf{H}(c_{mid}, c_{in}) \sum_{k=0}^{K-1} (\widetilde{\mathbf{F}}_{\mathbf{in}}(k, c_{in}) \mathbf{M}(k, c_{mid}))) \\ &= Conv_{1\times1}(\mathbf{H}, \widetilde{\mathbf{F}}_{\mathbf{in}}^T \mathbf{M}) \end{aligned} \tag{2.7}$$

Thus, the original PointConv can be equivalently reduced to a matrix multiplication and

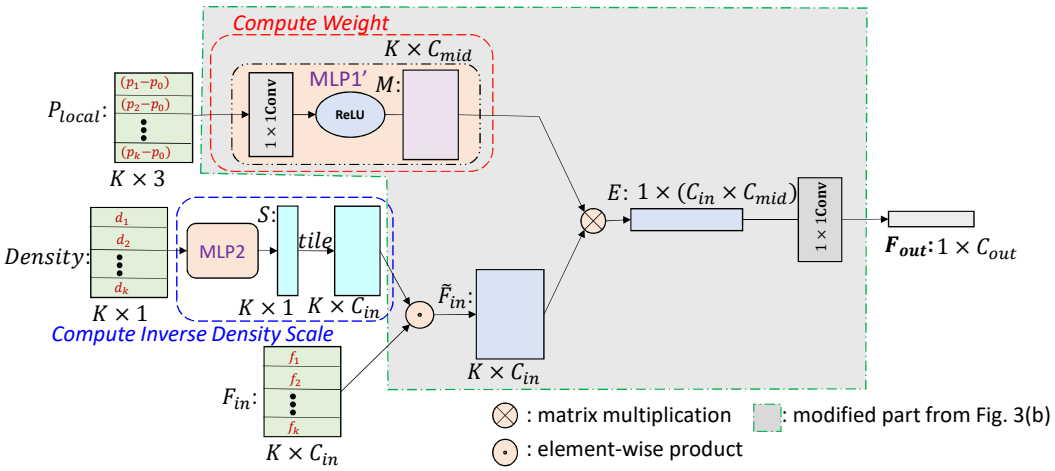


Figure 2.5: Efficient PointConv. The memory efficient version of PointConv on one local region with K points.

a 1×1 convolution. Figure 2.5 shows the efficient version of PointConv.

In this method, instead of storing the generated filters in memory, we divide the weights filters into two parts: the intermediate result M and the convolution kernel H. As we can see, the memory consumption reduces to $\frac{\mathbf{C_{mid}}}{\mathbf{K} \times \mathbf{C_{out}}}$ of the original version. With the same input setup as the Figure 2.3 and let $C_{mid} = 32$, the memory consumption is $0.1255GB$, which is about $1/64$ of the original PointConv.

2.5 Experiments

In order to evaluate our new PointConv network, we conduct experiments on several widely used datasets, ModelNet40 [160], ShapeNet [14] and ScanNet [23]. In order to demonstrate that our PointConv is able to fully approximate conventional convolution, we also report results on the CIFAR-10 dataset [64]. In all experiments, we implement the models with Tensorflow on a GTX 1080Ti GPU using the Adam optimizer. ReLU and batch normalization are applied after each layer except the last fully connected layer.

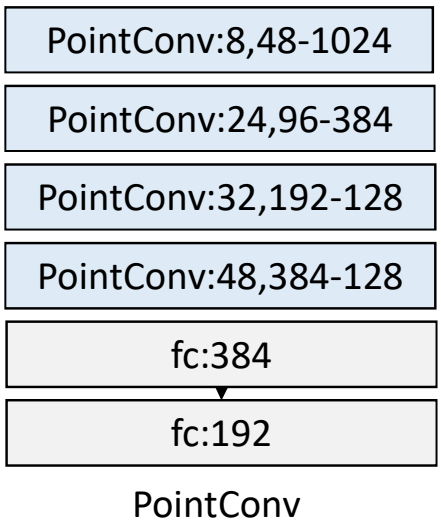


Figure 2.6: The network structures for ModelNet40 classification task. In the figure, $PointConv : 8, 48 - 1024$ is a PointConv layer with neighborhood size $K = 8$, $C_{out} = 48$ output channels, and $N' = 1024$ centroids.

2.5.1 Classification on ModelNet40

ModelNet40 contains 12,311 CAD models from 40 man-made object categories. We use the official split with 9,843 shapes for training and 2,468 for testing. Following the configuration in [106], we use the source code for PointNet [106] to sample 1,024 points uniformly and compute the normal vectors from the mesh models. For fair comparison, we employ the same data augmentation strategy as [106] by randomly rotating the point cloud along the z-axis and jittering each point by a Gaussian noise with zero mean and 0.02 standard deviation. The network structures we used in our experiment is shown in Figure 2.6. In Table 2.1, PointConv achieved state-of-the-art performance among methods based on 3D input. ECC[118] which is similar to our approach, cannot scale to a large network, which limited their performance.

2.5.2 ShapeNet Part Segmentation

Part segmentation is a challenging fine-grained 3D recognition task. The ShapeNet dataset contains 16,881 shapes from 16 classes and 50 parts in total. The input of the

Table 2.1: ModelNet40 Classification Accuracy.

Method	Input	Accuracy(%)
Subvolume [107]	voxels	89.2
ECC [118]	graphs	87.4
Kd-Network [62]	1024 points	91.8
PointNet [106]	1024 points	89.2
PointNet++ [108]	1024 points	90.2
PointNet++ [108]	5000 points+normal	91.9
SpiderCNN [163]	1024 points+normal	92.4
PointConv	1024 points+normal	**92.5**

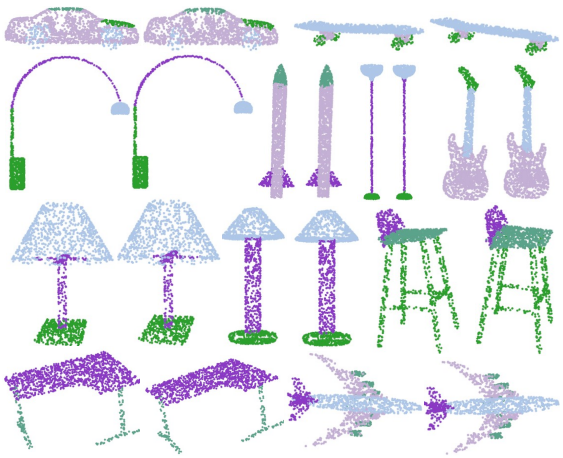


Figure 2.7: Part segmentation results. For each pair of objects, the left one is the ground truth, the right one is predicted by PointConv. Best viewed in color.

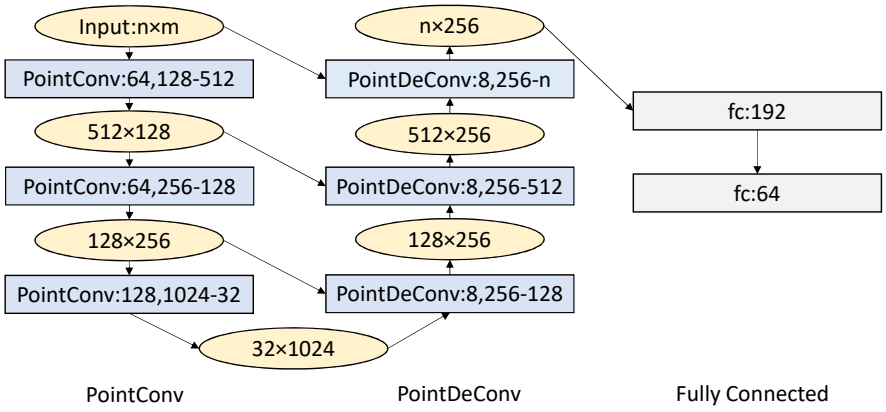


Figure 2.8: The network structure for ShapeNet part segmentation. *PointConv* : $64, 128 - 512$ is a PointConv layer with neighborhood size $K = 64$, $C_{out} = 128$ output channels, and $N = 512$ centroids. Each rectangle represents a convolution/deconvolution layer. Each ellipse represents the data dimensionality at the particular stage. 512×128 means the point cloud has 512 points with 128-dimensional features.

Table 2.2: Results on ShapeNet part dataset. Class avg. is the mean IoU averaged across all object categories, and inctance avg. is the mean IoU across all objects.

	airplane	bag	cap	car	chair	earphone	guitar	knife	lamp	laptop	motor	mug	pistol	rocket	skate board	table	class avg.	instance avg.
SSCNN [169]	81.6	81.7	81.9	75.2	90.2	74.9	93.0	86.1	**84.7**	95.6	66.7	92.7	81.6	60.6	**82.9**	82.1	82.0	84.7
Kd-net [62]	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3	77.4	82.3
PN [106]	83.4	78.7	82.5	74.9	89.6	73.0	**91.5**	85.9	80.8	95.3	65.3	93.0	81.2	57.9	72.8	80.6	80.4	83.7
PN++[108]	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6	81.9	85.1
SpiderCNN [163]	**83.5**	81.0	87.2	77.5	90.8	**76.8**	91.1	**87.3**	83.3	95.8	70.2	93.5	**82.7**	59.7	75.8	82.8	82.4	85.3
SPLATNet$_{3D}$ [123]	81.9	**83.9**	**88.6**	**79.5**	90.1	73.5	91.3	84.7	84.5	**96.3**	69.7	95.0	81.7	59.2	70.4	81.3	82.0	84.6
PointConv	83.5	80.9	87.5	**79.5**	**90.9**	75.8	90.9	86.6	84.6	95.8	**72.4**	**95.3**	81.8	**60.8**	75.2	**83.2**	**82.8**	**85.7**

task is shapes represented by a point cloud, and the goal is to assign a part category label to each point in the point cloud. The category label for each shape is given. We follow the experiment setup in most related work [108, 123, 163, 62]. It is common to narrow the possible part labels to the ones specific to the given object category by using the known input 3D object category. And we also compute the normal direction on each point as input features to better describe the underlying shape. Figure 2.7 visualizes some sample results. The network structure is shown in Figure 2.8.

We use point intersection-over-union(IoU) to evaluate our PointConv network, same as PointNet++ [108], SPLATNet [123] and some other part segmentation algorithms

[169, 62, 163, 40]. The results are shown in Table 2.2. PointConv obtains a class average mIoU of 82.8% and an instance average mIoU of 85.7%, which are on par with the state-of-the-art algorithms which only take point clouds as input. According to [123], the SPLATNet$_{2D-3D}$ also takes rendered 2D views as input. Since our PointConv only takes 3D point clouds as input, for fair comparison, we only compare our result with the SPLATNet$_{3D}$ in [123].

2.5.3 Semantic Scene Labeling

Datasets such as ModelNet40 [160] and ShapeNet [14] are man-made synthetic datasets. As we can see in the previous section, most state-of-the-art algorithms are able to obtain relatively good results on such datasets. To evaluate the capability of our approach in processing realistic point clouds, which contains a lot of noisy data, we evaluate our PointConv on semantic scene segmentation using the ScanNet dataset. The task is to predict semantic object labels on each 3D point given indoor scenes represented by point clouds. The newest version of ScanNet [23] includes updated annotations for all 1513 ScanNet scans and 100 new test scans with all semantic labels publicly unavailable and we submitted our results to the official evaluation server to compare against other approaches.

We compare our algorithm with Tangent Convolutions [133], SPLAT Net [123], PointNet++ [108] and ScanNet [23]. All the algorithm mentioned reported their results on the new ScanNet dataset to the benchmark, and the inputs of the algorithms only uses 3D coordinates data plus RGB. In our experiments, we generate training samples by randomly sample $3m \times 1.5m \times 1.5m$ cubes from the indoor rooms, and evaluate using a sliding window over the entire scan. We report intersection over union (IoU) as our main measures, which is the same as the benchmark. We visualize some example semantic segmentation results in Figure 2.9. The mIoU is reported in Table 2.3. The mIoU is the mean of IoU across all the categories. Our PointConv outperforms other algorithm by a significant margin (Table 2.3). The total running time of PointConv for training one epoch on ScanNet on one GTX1080Ti is around 170s, and the evaluation time with 8×8192 points is around 0.5s.

In Table 2.4, we report the IoU of each category on the ScanNet dataset. In all, we can

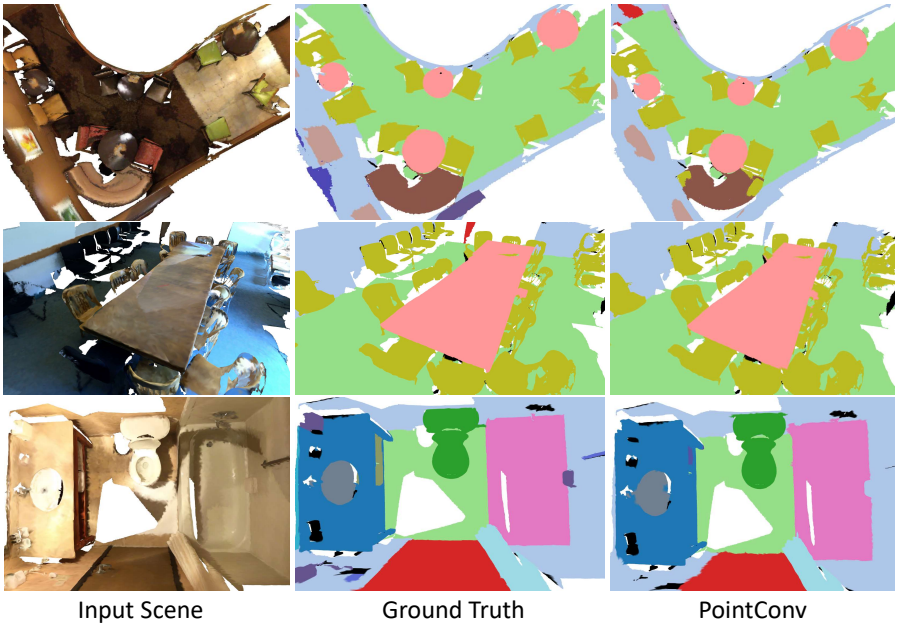


Figure 2.9: Examples of semantic scene labeling. The images from left to right are the input scenes, the ground truth segmentation, and the prediction from PointConv. For better visualization, the point clouds are converted into mesh format. Best viewed in color.

Table 2.3: Semantic Scene Segmentation results on ScanNet.

Method	mIoU(%)
ScanNet [23]	30.6
PointNet++ [108]	33.9
SPLAT Net [123]	39.3
Tangent Convolutions [133]	43.8
PointConv	**55.6**

Table 2.4: Segmentation results on Each Category in IoU (%).

shape	bathtub	bed	bookshelf	cabinet	chair	counter	curtain	desk	door	floor
ScanNet [23]	20.3	36.6	50.1	31.1	52.4	21.1	0.2	34.2	18.9	78.6
PointNet++ [108]	58.4	47.8	45.8	25.6	36.0	25.0	24.7	27.8	26.1	67.7
SPLATNet [123]	47.2	51.1	**60.6**	31.1	65.6	24.5	40.5	32.8	19.7	92.7
Tangent Concolution [133]	43.7	**64.6**	47.4	36.9	64.5	35.3	25.8	28.2	27.9	91.8
PointCNN [76]	51.0	58.3	41.7	41.4	70.8	24.1	36.7	40.5	32.3	**94.4**
PointConv	**63.6**	64.0	57.4	**47.2**	**73.9**	**43.0**	**43.3**	**41.8**	**44.5**	**94.4**
shape	otherfurniture	picture	refrigerator	shower curtain	sink	sofa	table	toilet	wall	window
ScanNet [23]	14.5	10.2	24.5	15.2	31.8	34.8	30.0	46.0	43.7	18.2
PointNet++ [108]	18.3	11.7	21.2	14.5	36.4	34.6	23.2	54.8	52.3	25.2
SPLAT Net [123]	22.7	0	0.1	24.9	27.1	51.0	38.3	59.3	69.9	26.7
Tangent Concolution [133]	29.8	14.7	28.3	29.4	48.7	56.2	42.7	61.9	63.3	35.2
PointCNN [76]	30.0	13.2	22.6	41.7	53.4	52.5	**51.1**	80.6	74.3	47.9
PointConv	**37.2**	**18.5**	**46.4**	**57.5**	**54.0**	**63.9**	50.5	**82.7**	**76.2**	**51.5**

see that our approach outperforms other algorithms in almost all categories. One can see that in some categories, PointConv significantly improves over others, such as *counter* (35.3 % $\rightarrow$ 43.0 %), *door* (32.3 % $\rightarrow$ 44.5 %), *refrigerator* (28.3 % $\rightarrow$ 46.4 %), *shower curtain* (41.7 % $\rightarrow$ 57.5 %), *sofa* (56.2 % $\rightarrow$ 63.9 %).

2.5.4 Classification on CIFAR-10

In Sec.2.3.1, we claimed that PointConv can be equivalent with 2D CNN. If this is true, then the performance of a network based on PointConv should be equivalent to that of a raster image CNN. In order to verify that, we use the CIFAR-10 dataset as a comparison benchmark. We treat each pixel in CIFAR-10 as a 2D point with xy coordinates and RGB features. The point clouds are scaled onto the unit ball before training and testing.

Experiments show that PointConv on CIFAR-10 indeed has the same learning capacities as a 2D CNN. Table 2.5 shows the results of image convolution and PointConv. From the table, we can see that the accuracy of PointCNN[76] on CIFAR-10 is only 80.22%, which is much worse than image CNN. However, for 5-layer networks, the network using PointConv is able to achieve 89.13%, which is similar to the network using image convolution. And, PointConv with VGG19 [119] structure can also achieve on par accuracy comparing with VGG19.

Table 2.5: CIFAR-10 Classification Accuracy.

	Accuracy(%)
Image Convolution	88.52
AlexNet [65]	89.00
VGG19 [119]	93.60
PointCNN [76]	80.22
SpiderCNN [163]	77.97
PointConv(5-layer)	89.13
PointConv(VGG19)	93.19

2.6 Ablation Experiments and Visualizations

In this section, we conduct additional experiments to evaluate the effectiveness of each aspect of PointConv. Besides the ablation study on the structure of the PointConv, we also give an in-depth breakdown on the performance of PointConv on the ScanNet dataset. Finally, we provide some learned filters for visualization.

2.6.1 The Structure of MLP

In this section, we design experiments to evaluate the choice of MLP parameters in PointConv. For fast evaluation, we generate a subset from the ScanNet dataset as a classification task. Each example in the subset is randomly sampled from the original scene scans with 1,024 points. There are 20 different scene types for the ScanNet dataset. The reason why we use a subset of ScanNet dataset is we want to avoid fitting the wrong parameters on a dataset that is too simple such as ModelNet40. The selected dataset is a realistic 3D point cloud with RGB information. Since the problem is complex enough, we could imagine parameters that are good enough for other datasets with similar complexity.

We empirically sweep over different choices of C_{mid} and different number of layers of the MLP in PointConv. Each experiment was conducted for 3 random trials. The results is shown in Figure 2.10. From the results, we find that larger C_{mid} does not necessarily give better classification results. And the different number of layers in MLP does not give much difference in classification results. Since C_{mid} is linearly correlated with the

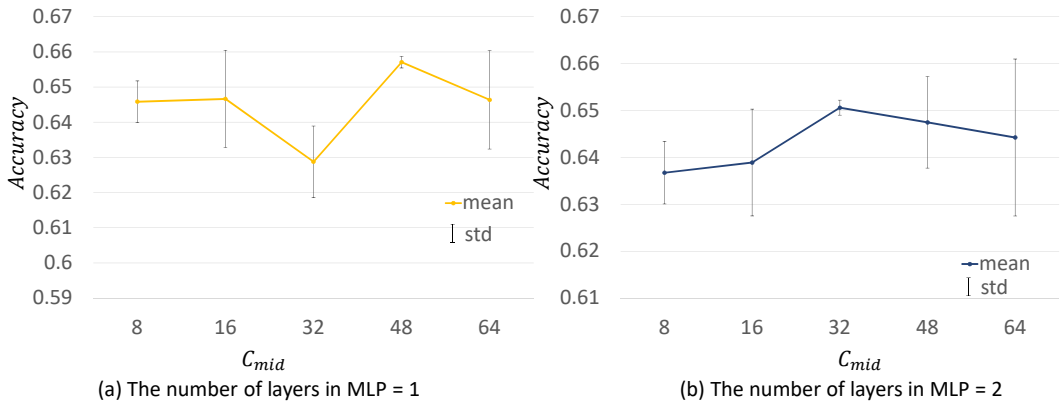


Figure 2.10: Classification accuracy of different choice of C_{mid} and layers number of MLP.

memory consumption of each PointConv layer, this results shows that we can choose a reasonably small C_{mid} for greater memory efficiency.

2.6.2 Inverse Density Scale

In this section, we study the effectiveness of the inverse density scale S. We choose ScanNet as our evaluation task since the point clouds in ScanNet are generated from real indoor scenes. We follow the standard training/validation split provided by the authors. We train the network with and without the inverse density scale as described in Sec. 2.3.1, respectively. Table 2.6 shows the results. As we can see, PointConv with inverse density scale performs better than the one without by about 1%, which proves the effectiveness of inverse density scale. In our experiments, we observe that inverse density scale tend to be more effective in layers closer to the input. In deep layers, the MLP tends to diminish the effect of the density scale. One possible reason is that with farthest point sampling algorithm as our sub-sampling algorithm, the point cloud in deeper layer tend to be more uniformly distributed. And as shown in Table 2.6, directly applying density without using the nonlinear transformation gives worse result comparing with the one without density on ScanNet dataset, which shows that the nonlinear transform is able to learn the inverse density scale in the dataset.

2.6.3 Ablation Studies on ScanNet

As one can see, our PointConv outperforms other approaches with a large margin. Since we are only allowed to submit one final result of our algorithm to the benchmark server of ScanNet, we perform more ablation studies for PointConv using the public validation set provide by [23]. For the segmentation task, we train our PointConv with 8,192 points randomly sampled from a $3m \times 1.5m \times 1.5m$, and evaluate the model with exhaustively choose all points in the $3m \times 1.5m \times 1.5m$ cube in a sliding window fashion through the xy-plane with different stride sizes. For robustness, we use a majority vote from 5 windows in all of our experiments. From Table 2.6, we can see that smaller stride size is able to improve the segmentation results, and the RGB information on ScanNet does not seem to significantly improve the segmentation results. Even without these additional improvements, PointConv still outperforms baselines by a large margin.

Table 2.6: Ablation study on ScanNet. With and without RGB information, inverse density scale and using different stride size of sliding window.

Input	Stride Size(m)	mIoU	mIoU/No Density	mIoU/Density(no MLP)
xyz	0.5	61.0	60.3	60.1
	1.0	59.0	58.2	57.7
	1.5	58.2	56.9	57.3
xyz+RGB	0.5	60.8	58.9	-
	1.0	58.6	56.7	-
	1.5	57.5	56.1	-

2.6.4 Visualization

Continuous Filters. Figure 2.11 visualizes the learned filters from the MLPs in our PointConv. In order to better visualize the filters, we sample the learned functions through a plane $z = 0$. From the Figure 2.11, we can see some patterns in the learned continuous filters.

ShapeNet Part Segmentation. We visualize more segmentation results for ShapeNet part segmentation in Figure 2.12.

ScanNet Semantic Segmentation. We visualize more segmentation results for Scan-

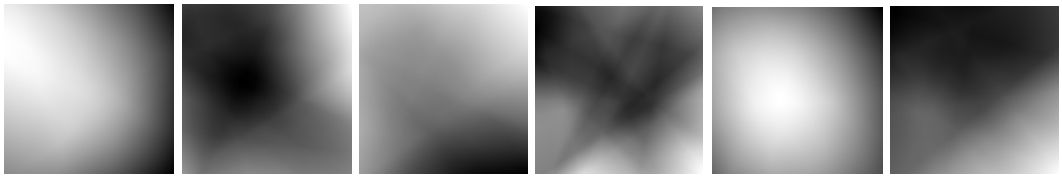


Figure 2.11: Learned Convolutional Filters. The convolution filters learned by the MLPs on ShapeNet.For better visualization, we take all weights filters from $z = 0$ plane.

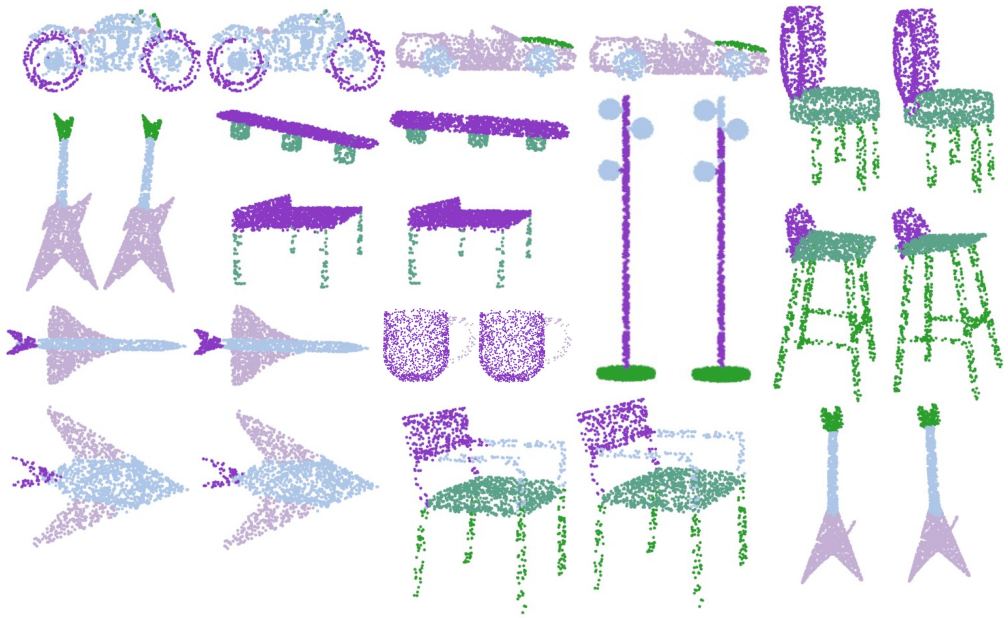


Figure 2.12: Part segmentation results. For each pair of objects, the left one is the ground truth, the right one is predicted by PointConv. Best viewed in color.

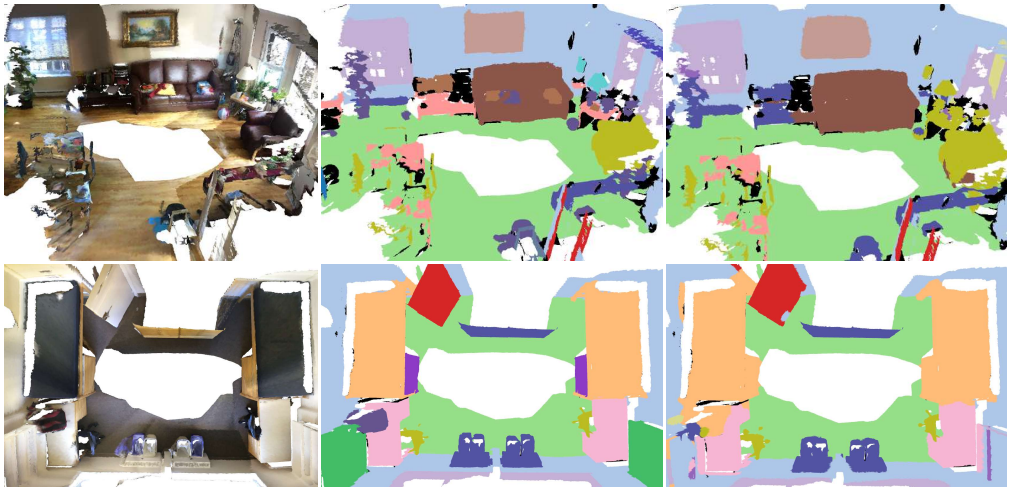


Figure 2.13: Examples of semantic scene labeling. The images from left to right are the input scenes, the ground truth segmentation, and the prediction from PointConv. For better visualization, the point clouds are converted into mesh format. Best viewed in color.

Net semantic segmentation in Figure 2.13, 2.14, and 2.15.

2.7 Conclusion

In this work, we proposed a novel approach to perform convolution operation on 3D point clouds, called PointConv. PointConv trains multi-layer perceptrons on local point coordinates to approximate continuous weight and density functions in convolutional filters, which makes it naturally permutation-invariant and translation-invariant. This allows deep convolutional networks to be built directly on 3D point clouds. We proposed an efficient implementation of it which greatly improved its scalability. We demonstrated its strong performance on multiple challenging benchmarks and capability of matching the performance of a grid-based convolutional network in 2D images. In future work, we would like to adopt more mainstream image convolution network architectures into point cloud data using PointConv, such as ResNet and DenseNet. The code can be found here: https://github.com/DylanWusee/pointconv.

Figure 2.14: Examples of semantic scene labeling. The images from left to right are the input scenes, the ground truth segmentation, and the prediction from PointConv. For better visualization, the point clouds are converted into mesh format. Best viewed in color.

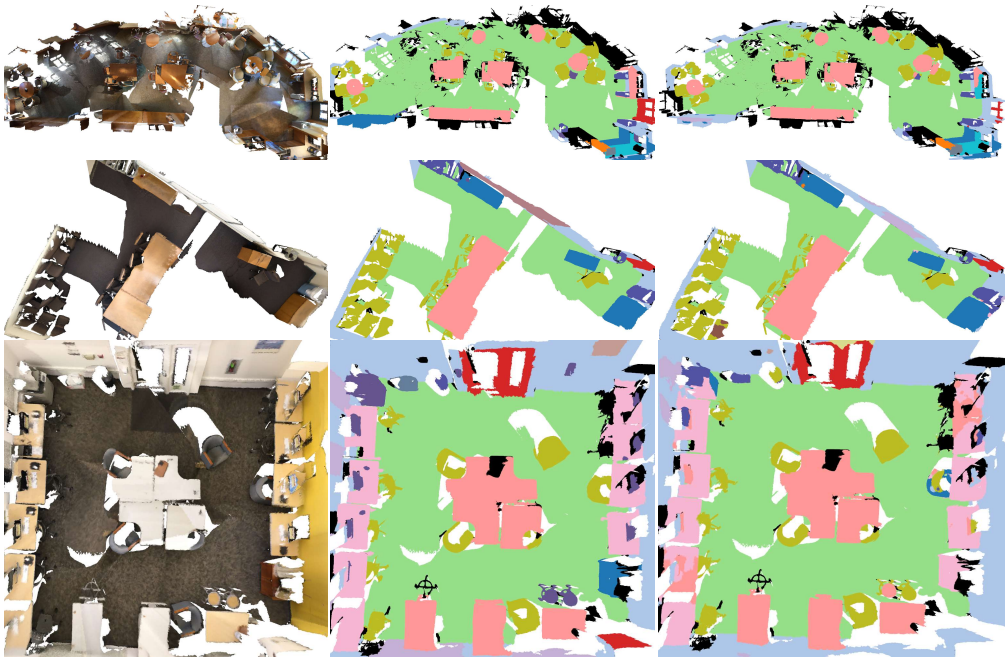


Figure 2.15: Examples of semantic scene labeling. The images from left to right are the input scenes, the ground truth segmentation, and the prediction from PointConv. For better visualization, the point clouds are converted into mesh format. Best viewed in color.

Chapter 3: Cost Volume on Point Clouds for (Self-)Supervised Scene Flow Estimation

3.1 Introduction

Scene flow is the 3D displacement vector between each surface point in two consecutive frames. As a fundamental tool for low-level understanding of the world, scene flow can be used in various applications, such as motion segmentation, action recognition, autonomous driving, etc. Traditionally, scene flow was estimated directly from RGB data [93, 91, 143, 145]. But recently, due to the increasing application of 3D sensors such as LiDAR, there is interest on directly estimating scene flow from 3D point clouds.

Fueled by recent advances in 3D deep networks that learn effective feature representations directly from point cloud data, recent work adopt ideas from 2D deep optical flow networks to 3D to estimate scene flow from point clouds. FlowNet3D [78] operates directly on points with PointNet++ [108], and proposes a *flow embedding* which is computed in one layer to capture the correlation between two point clouds, and then propagates it through finer layers to estimate the scene flow. HPLFlowNet [42] computes the correlation jointly from multiple scales utilizing the upsampling operation in bilateral convolutional layers.

An important piece in deep optical flow estimation networks is the cost volume [61, 162, 127], a 3D tensor that contains matching information between neighboring pixel pairs from consecutive frames. In this paper, we propose a novel learnable point-based cost volume where we discretize the cost volume to input point pairs, avoiding the creation of a dense 4D tensor if we naively extend from the image to point cloud. Then we apply the efficient PointConv layer [158] on this irregularly discretized cost volume. We experimentally show that it outperforms previous approaches for associating point cloud correspondences, as well as the cost volume used in 2D optical flow. We also propose efficient upsampling and warping layers to implement a coarse-to-fine flow estimation

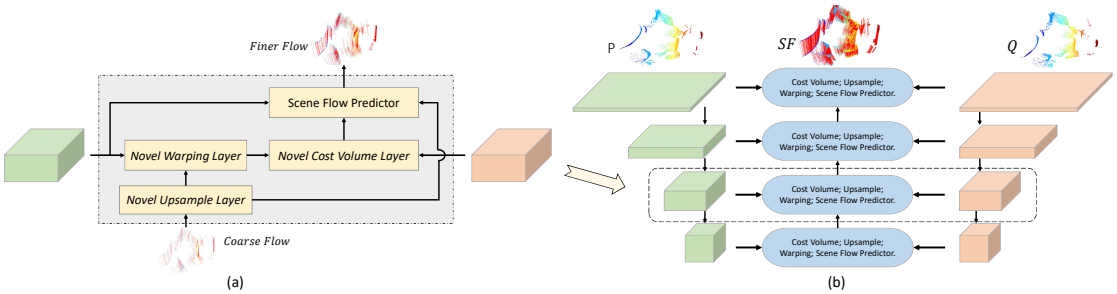


Figure 3.1: (a) illustrates how the pyramid features are used by the novel cost volume, warping, and upsample layers in one level. (b) shows the overview structure of PointPWC-Net. At each level, PointPWC-Net first warps the features from the first point cloud using the upsampled scene flow. Then, the cost volume is computed using the feature from the warped first point cloud and the second point cloud. Finally, the scene flow predictor predicts finer flow at the current level using features from the first point cloud, the cost volume, and the upsampled flow. (Best viewed in color)

framework.

As in optical flow, it is difficult and expensive to acquire accurate scene flow labels for point clouds. Hence, beyond supervised scene flow estimation, we also explore self-supervised scene flow which does not require human annotations. *To our knowledge, our work is the first to explore deep self-supervised scene flow estimation from point cloud data.* We propose new self-supervised loss terms: Chamfer distance [32], smoothness constraint and Laplacian regularization. These loss terms enable us to achieve state-of-the-art performance without any supervision.

We conduct extensive experiments on FlyingThings3D [91] and KITTI Scene Flow 2015 [97, 95] datasets with both supervised loss and the proposed self-supervised losses. Experiments show that the proposed PointPWC-Net outperforms all the previous methods with a large margin. The self-supervised version is comparable with some of the previous supervised methods on FlyingThings3D, such as SPLATFlowNet [123]. On KITTI where supervision is not available, our self-supervised version achieves better performance than the supervised version trained on FlyingThings3D, far surpassing state-of-the-art. We also ablate each critical component of PointPWC-Net to understand their contributions.

The key contributions of our work are:

• We propose a novel learnable cost volume layer that performs convolution on the cost volume without creating a dense 4-dimensional tensor.

• We present a novel model, called PointPWC-Net, that estimates scene flow from two consecutive point clouds in a coarse-to-fine fashion with the help of the novel learnable cost volume layer.

• We introduce self-supervised losses that can train the PointPWC-Net without any ground truth label. To our knowledge, we are the first to propose such idea in 3D point cloud deep scene flow estimation.

• We achieve state-of-the-art performance on FlyingThing3D and KITTI Scene Flow 2015, far surpassing previous state-of-the-art.

3.2 Related Work

Deep Learning on Point Clouds. Deep learning methods on 3D point clouds have gained more and more attention in the past several years. Some latest work [112, 106, 108, 123, 133, 52, 41, 144, 76] directly take raw point clouds as input. [112, 106, 108] use a shared multi-layer perceptron (MLP) and max pooling layer to obtain features of point clouds. SPLATNet [123] projects the input features of the point clouds onto a high-dimensional lattice, and then apply bilateral convolution on the high-dimensional lattice to aggregate features. Other work [118, 59, 155, 45, 151, 158] propose to learn continuous convolutional filter weights as a nonlinear function from 3D point coordinates, approximated with MLP. [45, 158] use a density estimation to compensate the non-uniform sampling, and [158] significantly improves the memory efficiency by a change of summation trick, allowing these networks to scale up and achieving comparable capabilities with 2D convolution.

Optical Flow Estimation. Optical flow estimation is a core computer vision problem and has many applications. Traditionally, the top performing methods adopt the energy minimization approach [47] and a coarse-to-fine, warping-based method [8, 12, 11]. Since FlowNet [30], there were many recent breakthroughs using a deep network to learn optical flow. [56] stacks several FlowNet into a larger one. [111] develops a compact spatial pyramid network. [127] integrates the widely used traditional pyramid, warping,

and cost volume technique into CNNs for optical flow, and outperform all the previous methods with high efficiency. We utilized a basic structure similar to theirs but proposed novel cost volume, warping and upsample layers appropriate for point clouds.

Scene Flow Estimation. 3D scene flow is first introduced by [143]. Many works [53, 93, 146] estimate scene flow using RGB data. [53] introduces a variational method to estimate scene flow from stereo sequences. [93] proposes an object-level scene flow estimation approach and introduces a dataset for 3D scene flow. [146] presents a piecewise rigid scene model for 3D scene flow estimation.

Recently, there are some works [28, 141, 140] that estimate scene flow directly from point clouds using classical techniques. [28] introduces a method that formulates the scene flow estimation problem as an energy minimization problem with assumptions on local geometric constancy and regularization for motion smoothness. [141] proposes a real-time four-steps method of constructing occupancy grids, filtering the background, solving an energy minimization problem, and refining with a filtering framework. [140] further improves the method in [141] by using an encoding network to learn features from an occupancy grid.

In some most recent work [151, 78, 42], researchers try to estimate scene flow from point clouds using deep learning in a end-to-end fashion. [151] uses PCNN to operate on LiDAR data to estimate LiDAR motion. [78] introduces FlowNet3D based on PointNet++ [108]. FlowNet3D uses a flow embedding layer to encode the motion of point clouds. However, it requires encoding a large neighborhood in order to capture large motions. [42] presents HPLFlowNet to estimate the scene flow using Bilateral Convolutional Layers(BCL), which projects the point cloud onto a permutohedral lattice. [5] estimates scene flow by using a network that jointly predicts 3D bounding boxes and rigid motions of objects or background in the scene. Different from [5], we do not require the rigid motion assumption and segmentation level supervision to estimate scene flow.

Self-supervised Scene Flow. There are several recent works [77, 170, 183, 69] which jointly estimate multiple tasks, i.e. depth, optical flow, ego-motion and camera pose without supervision. They take 2D images as input, which have ambiguity when used in scene flow estimation. In this paper, we investigate self-supervised learning of scene flow from 3D point clouds with our PointPWC-Net. To the best of our knowledge, we are

the first to study self-supervised deep learning of scene flow from 3D point clouds.

Traditional Point Cloud Registration. Point cloud registration has been extensively studied well before deep learning [43, 130]. Most of the work [19, 37, 46, 87, 98, 117, 135] consist two stages: global alignment followed by local refinement. This only works when most of the motion in the scene is globally rigid. Many methods are based on the iterative closest point(ICP) [9] and its variants [104]. [180] introduces fast global registration (FGR) that significantly improves the efficiency of the registration process. However, real-world point clouds are usually deform in a non-rigid way, which are much more difficult for registration [130]. Several works [1, 99, 57, 10] deal with non-rigid point cloud registration. [99] introduces a probabilistic method, called Coherent Point Drift(CPD), for both rigid and non-rigid point set registration. However, the computation overhead makes it hard to apply on real world data in real-time. Many algorithms are proposed to extend the CPD method [84, 100, 81, 181, 35, 85, 33, 2, 34, 83, 82, 70, 132, 115, 25, 80, 174, 148, 171, 109]. In [100], the algorithm uses multiple kernel functions for motion estimation. [181] replaces Gaussian distribution with the Student′s-t distribution.In [84], some local structure descriptors[35, 85] are proposed to preserve the local structure of point sets in registration. Some algorithms require additional information for point set registration. The work [115, 25] takes the color information along with the spatial location into account. [1] requires meshes for non-rigid registration. In [109], the regression and clustering for point set registration in a Bayesian framework are presented. All the aforementioned work require optimization at inference time, which has significantly higher computation cost than our method which run in a fraction of a second for inference.

3.3 PointPWC-Net

To compute optical flow with high accuracy, one of the most important components is the cost volume. In 2D images, the cost volume can be computed by aggregating the cost in a square neighborhood on a grid. However, computing cost volume across two point clouds is difficult since 3D point clouds are unordered with a nonuniform sampling density. In this section, we introduce a novel learnable cost volume layer, and use it to construct a deep network with the help of other auxiliary layers that outputs high

quality scene flow.

3.3.1 The Cost Volume Layer

As one of the key components of optical flow estimation, most state-of-the-art algorithms, both traditional [126, 113] and modern deep learning based ones [127, 162, 13], use the cost volume to estimate optical flow. However, computing cost volumes on point clouds is still an open-problem. There are several works that compute some kind of flow embedding or correlation between point clouds. It is still hard to construct a real cost volume in point clouds. One of the most straightforward way of computing cost volume is computing the feature correlation followed by several CNN layers, as in [127]. This method is problematic due to the permutation invariance of point clouds. [78] proposes a flow embedding layer to aggregate feature similarities and spatial relationships to encode point motions. However, the motion information between points can be lost due to the max pooling operation in the flow embedding layering. [42] introduces a CorrBCL layer to compute the correlation between two point clouds, which requires to transfer two point clouds onto the same permutohedral lattice.

To address these issues, we present a novel learnable cost volume layer that directly applies onto the features of two point clouds. Suppose $f_i \in \mathbb{R}^c$ is the feature for point $p_i \in P$ and $g_j \in \mathbb{R}^c$ the feature for point $q_j \in Q$, the matching cost between p_i and q_j can be defined as:

$$Cost(p_i, q_j) = h(f_i, g_j, q_j, p_i) \tag{3.1}$$

$$= MLP(concat(f_i, g_j, q_j - p_i)) \tag{3.2}$$

In our network, the feature f_i and g_j are either the raw coordinates of the point clouds, or the convolution output from previous layers. The intuition is that, as a universal approximator, MLP should be able to learn the potentially nonlinear relationship between the two points. Due to the flexibility of the point cloud, we also add a direction vector $(q_j - p_i)$ to the computation besides the point features f_i and g_j.

Once we have the matching costs, they can be aggregated as a cost volume for predicting

the movement between two point clouds. In 2D images, aggregating the cost is simply by applying some convolutional layers as in PWC-Net [127]. However, the traditional convolutional layers can not be applied directly on point clouds due to the unorderness. Besides, [78] uses max-pooing to aggregate features in the second point cloud. [42] uses CorrBCL to aggregate features on a permutohedral lattice. However, their methods only aggregate costs in a point-to-point manner, which is sensitive to outliers. To obtain robust and stable cost volumes, in this work, we propose to aggregate costs in a patch-to-patch manner similar to the cost volumes on 2D images [61, 127].

For a point p_c in P, we first find a neighborhood $N_P(p_c)$ around p_c in P. For each point $p_i \in N_P(p_c)$, we find a neighborhood $N_Q(p_i)$ around p_i in Q. The cost volume for p_c is defined as:

$$CV(p_c) = \sum_{p_i \in N_P(p_c)} W_P(p_i, p_c) \sum_{q_j \in N_Q(p_i)} W_Q(q_j, p_i)\ h(f_i, g_j, q_j, p_i) \tag{3.3}$$

$$h(f_i, g_j, q_j, p_i) = MLP(concat(f_i, g_j, q_j - p_i)) \tag{3.4}$$

$$W_P(p_i, p_c) = MLP(p_i - p_c) \tag{3.5}$$

$$W_Q(q_j, p_i) = MLP(q_j - p_i) \tag{3.6}$$

Where $W_P(p_i, p_c)$ and $W_Q(q_j, p_i)$ are the convolutional weights $w.r.t$ the direction vectors that are used to aggregate the costs from the patches in P and Q. It is learned as a continuous function of the directional vectors $(q_i - p_c) \in \mathbb{R}^3$ and $(q_j - p_i) \in \mathbb{R}^3$, respectively with an MLP, similar to PointConv [158] and PCNN [151]. The output of the cost volume layer is a tensor with shape (n_1, D), where n_1 is the number of points in P, and D is the dimension of the cost volume, which encodes all the motion information for each point. The patch-to-patch idea used in the cost volume is illustrated in Fig. 3.2.

There are two major differences between this cost volume for scene flow of 3D point clouds and conventional 2D cost volumes for stereo and optical flow. The first one is that we introduce a learnable function $h(\cdot) = MLP(concat(\cdot))$ that can dynamically learn the cost or correlation within the point cloud structures. Ablation studies in Sec.3.5.3 show that this novel learnable design achieve better results than traditional cost volume [127] in scene flow estimation. The second one is that this cost volume is

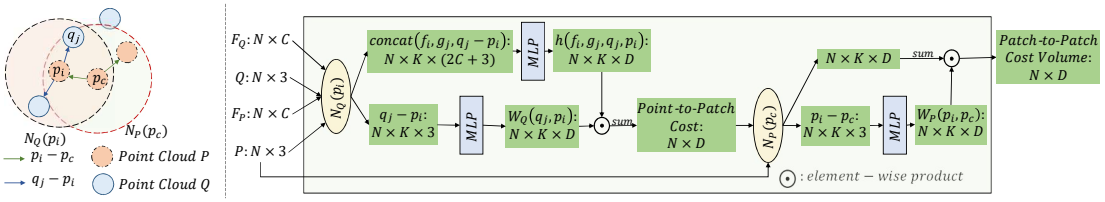


Figure 3.2: (a) Grouping. For a point p_c, we form its K-NN neighborhoods in each point cloud as $N_P(p_c)$ and $N_Q(p_c)$ for cost volume aggregation. We first aggregate the cost from the patch $N_Q(p_c)$ in point cloud Q. Then, we aggregate the cost from patch $N_P(p_c)$ in the point cloud P. (b) Cost Volume Layer. The features of neighboring points in $N_Q(p_c)$ are concatenated with the direction vector $(q_i - p_c)$ to learn a point-to-patch cost between p_c and Q with PointConv. Then the point-to-patch costs in $N_P(p_c)$ are further aggregated with PointConv to construct a patch-to-patch Cost Volume

discretized irregularly on the two input point clouds and their costs are aggregated with point-based convolution. Previously, in order to compute the cost volume for optical flow in a $d \times d$ area on a $W \times H$ 2D image, all the values in a $d^2 \times W \times H$ tensor needs to be populated, which is already slow to compute in 2D, but would be prohibitively costly in the 3D space. With (volumetric) 3D convolution, one needs to search a d^3 area to get a cost volume in 3D space. Our cost volume discretizes on input points and avoids this costly operation, while essentially creating the same capabilities to perform convolutions on the cost volume. With the proposed cost volume layer, we only need to find two neighborhoods $N_P(p_c)$ and $N_Q(p_i)$ of size K, which is much cheaper and does not depend on the number of points in a point cloud. In our experiments, we fix $|N_P(p_c)| = |N_Q(p_i)| = 16$. If a larger neighborhood is needed, we could subsample the neighborhood which would bring it back to the same speed. This subsampling operation is only applicable to the sparse point cloud convolution and not possible for conventional volumetric convolutions. We anticipate this novel cost volume layer to be widely useful beyond scene flow estimation. Table 3.2 shows that it is better than [78]'s MLP+Maxpool strategy.

3.3.2 Building Blocks of PointPWC-Net

Given the proposed learnable cost volume layer, it would be interesting to construct a deep network for scene flow estimation. As demonstrated in 2D optical flow estimation, one of the most effective methods for dense estimation is the coarse-to-fine structure. In this section, we introduce some novel auxiliary layers for point clouds that construct a coarse-to-fine network for scene flow estimation along with the proposed learnable cost volume layer. The network is called *"PointPWC-Net"* following [127].

As shown in Fig.3.1, PointPWC-Net predicts dense scene flow in a coarse-to-fine fashion. The input to PointPWC-Net is two consecutive point clouds, $P = \{p_i \in \mathbb{R}^3\}_{i=1}^{n_1}$ with n_1 points, and $Q = \{q_j \in \mathbb{R}^3\}_{j=1}^{n_2}$ with n_2 points. We first construct a feature pyramid for each point cloud. Afterwards, we build a cost volume using features from both point clouds at each layer. Then, we use the feature from P, the cost volume, and the upsampled flow to estimate the finer scene flow. We take the predicted scene flow as the coarse flow, upsample it to a finer flow, and warp points from P onto Q. Note that both the upsampling and the warping layers are efficient with no learnable parameters.

Feature Pyramid from Point Cloud. To estimate scene flow with high accuracy, we need to extract strong features from the input point clouds. We generate a L-level pyramid of feature representations, with the top level being the input point clouds, i.e., $l_0 = P/Q$. For each level l, we use furthest point sampling [108] to downsample the points by factor of 4 from previous level $l-1$, and use PointConv [158] to perform convolution on the features from level $l-1$. As a result, we can generate a feature pyramid with L levels for each input point cloud. After this, we enlarge the receptive field at level l of the pyramid by upsampling the feature in level $l+1$ and concatenate it to the feature at level l. Fig.3.3 shows the architecture for feature pyramid network.

Upsampling Layer. The upsampling layer can propagate the scene flow estimated from a coarse layer to a finer layer. We use a distance based interpolation to upsample the coarse flow. Let P^l be the point cloud at level l, SF^l be the estimated scene flow at level l, and p^{l-1} be the point cloud at level $l-1$. For each point p_i^{l-1} in the finer level point cloud P^{l-1}, we can find its K nearest neighbors $N(p_i^{l-1})$ in its coarser level point cloud P^l. The interpolated scene flow of finer level SF^{l-1} is computed using inverse distance weighted interpolation:

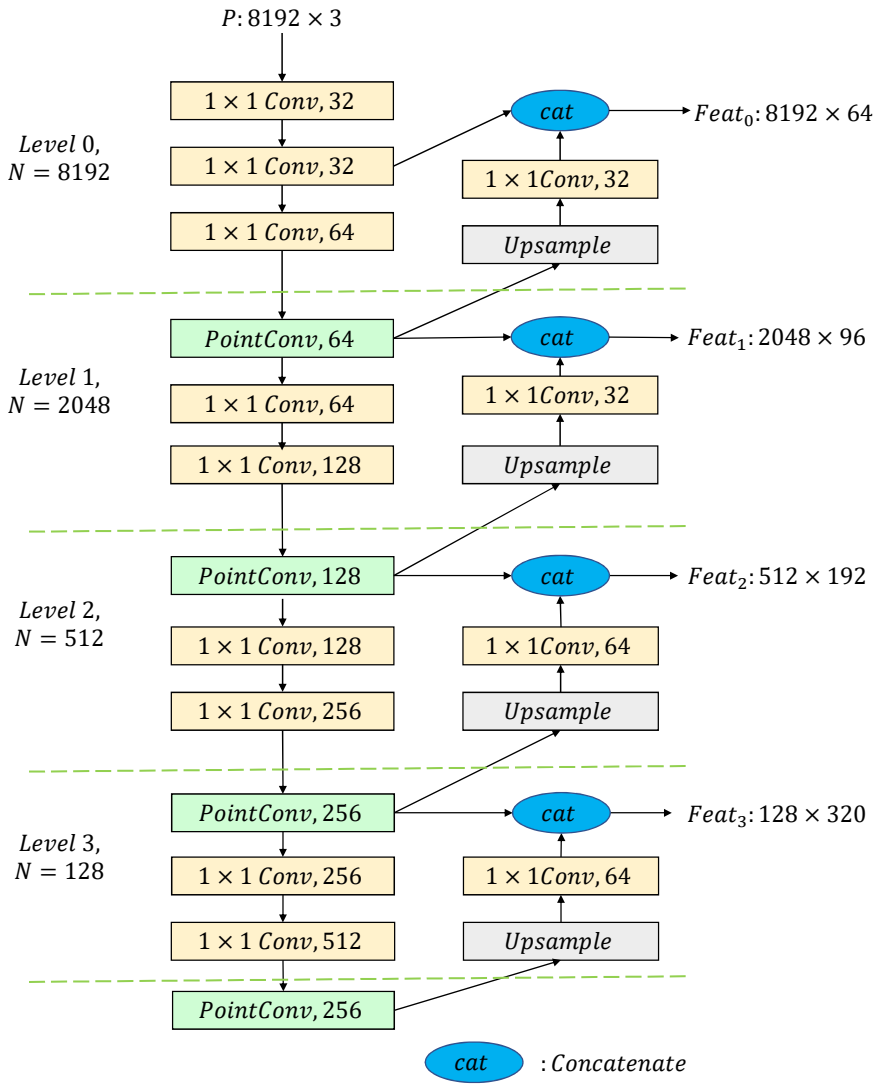


Figure 3.3: Feature Pyramid Network. The first point cloud and the second point cloud are encoded using the same network with shared weights. For each point cloud, we use PointConv [158] to convolve and downsample by factor of 4. The $1 \times 1 Conv$s are used to increase the representation power and efficiency. The final feature of level l is concatenated with the upsampled feature from level $l+1$, which contains feature with a larger receptive field.

$$SF^{l-1}(p_i) = \frac{\sum_{j=1}^{k} w(p_i^{l-1}, p_j^l) SF^l(p_j^l)}{\sum_{j=1}^{k} w(p_i^{l-1}, p_j^l)} \tag{3.7}$$

where $w(p_i^{l-1}, p_j^l) = 1/d(p_i^{l-1}, p_j^l)$, $p_i^{l-1} \in P^{l-1}$, and $p_j^l \in N(p_i^{l-1})$. $d(p_i^{l-1}, p_j^l)$ is a distance metric. We use Euclidean distance in this work.

Warping Layer. Warping would "apply" the computed flow so that only the residual flow needs to be estimated afterwards, hence the search radius can be smaller when constructing the cost volume. In our network, we first up-sample the scene flow from the previous coarser level and then warp it before computing the cost volume. Denote the upsampled scene flow as $SF = \{sf_i \in \mathbb{R}^3\}_{i=1}^{n_1}$, and the warped point cloud as $P_w = \{p_{w,i} \in \mathbb{R}^3\}_{i=1}^{n_1}$. The warping layer is simply an element-wise addition between the upsampled and computed scene flow $P_w = \{p_{w,i} = p_i + sf_i | p_i \in P, sf_i \in SF\}_{i=1}^{n_1}$. A similar warping operation is used for visualization to compare the estimated flow with the ground truth in [78, 42], but not used in coarse-to-fine estimation. [42] uses an offset strategy to reduce search radius which is specific to the permutohedral lattice.

Scene Flow Predictor. In order to obtain a flow estimate at each level, a convolutional scene flow predictor is built as multiple layers of PointConv and MLP. The inputs of the flow predictor are the cost volume, the feature of the first point cloud, the up-sampled flow from previous layer and the up-sampled feature of the second last layer from previous level's scene flow predictor, which we call the predictor feature. The intuition of adding predictor feature from coarse level is that predictor feature encodes all the information needed to predict scene flow at coarse level. By adding that, we might be able to correct a prediction with large error and improve robustness. The output is the scene flow $SF = \{sf_i \in \mathbb{R}^3\}_{i=1}^{n_1}$ of the first point cloud P. The first several PointConv layers are used to merge the feature locally, and the following MLP is used to estimate the scene flow on each point. We keep the flow predictor structure at different levels the same, but the parameters are not shared. Fig.3.4 shows the scene flow predictor network.

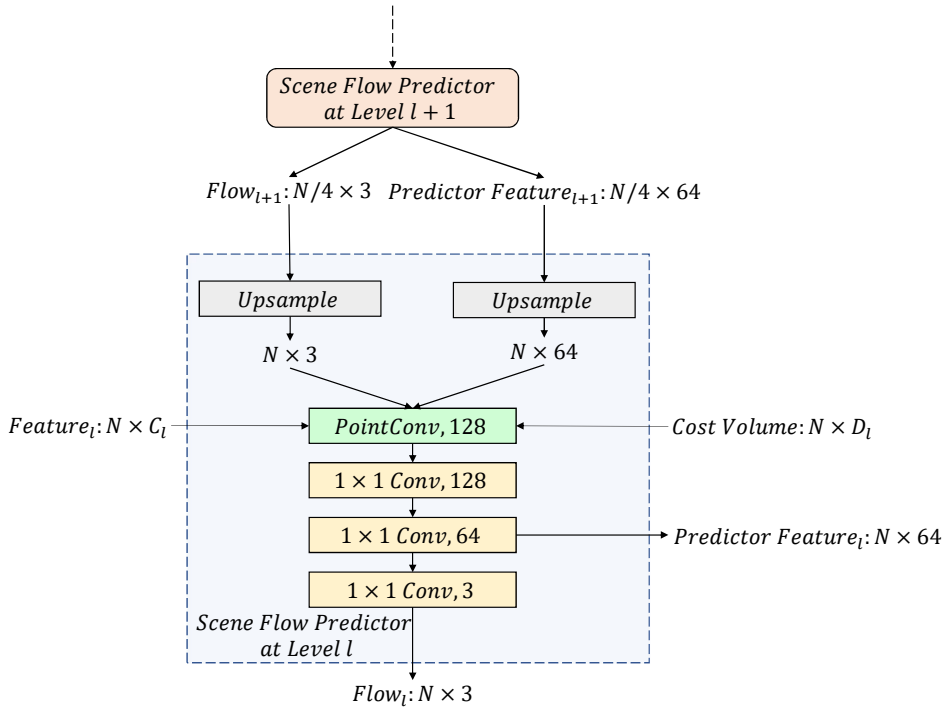


Figure 3.4: Scene Flow Predictor. The scene flow predictor takes the feature from the first point cloud, the cost volume, the upsampled flow from previous layer, and the upsampled feature of the second last layer from previous level's scene flow predictor as input. The output is the estimated flow in current level and the feature in the second last layer.

3.4 Training Loss Functions

In this section, we introduce two loss functions to train PointPWC-Net for scene flow estimation. One is the standard multi-scale supervised training loss, which has been explored in deep optical flow estimation [127] in 2D images. We use this supervised loss to train the model for fair comparison with previous scene flow estimation works, including FlowNet3D [78] and HPLFlowNet [42]. Due to that acquiring dense labeled 3D scene flow dataset is extremely hard, we also propose a novel self-supervised loss to train our PointPWC-Net without any supervision.

3.4.1 Supervised Loss

We adopt the multi-scale loss function in FlowNet [30] and PWC-Net [127] as a supervised learning loss to demonstrate the effectiveness of the network structure and the design choice. Let SF^l_{GT} be the ground truth flow at the l-th level. The multi-scale training loss $\boldsymbol{\ell}(\Theta) = \sum_{l=l_0}^{L} \alpha_l \sum_{p \in P} \left\| SF^l_\Theta(p) - SF^l_{GT}(p) \right\|_2$ is used where $\|\cdot\|_2$ computes the L_2-norm, α_l is the weight for each pyramid level l, and Θ is the set of all the learnable parameters in our PointPWC-Net, including the feature extractor, cost volume layer and scene flow predictor at different pyramid levels. Note that the flow loss is not squared as in [127] for robustness.

3.4.2 Self-supervised Loss

Obtaining the ground truth scene flow for 3D point clouds is difficult and there are not many publicly available datasets for scene flow learning from point clouds. Hence it would be interesting to investigate a self-supervised deep learning approach for scene flow from 3D point clouds. In this section, we propose a self-supervised learning objective function to learn the scene flow in 3D point clouds without supervision. Our loss function contains three parts: *Chamfer distance*, *Smoothness constraint*, and *Laplacian regularization* [150, 121]. To the best of our knowledge, we are the first to study self-supervised deep learning of scene flow estimation from 3D point clouds.

Chamfer Distance. The goal of using Chamfer loss is to estimate a scene flow by moving the first point cloud as close as the second one. Let SF^l_Θ be the scene flow

predicted at level l. Let P_w^l be the point cloud warped from the first point cloud P^l according to SF_Θ^l in level l, Q^l be the second point cloud at level l. Let p_w^l and q^l be points in P_w^l and Q^l. The Chamfer loss ℓ_C^l can be written as:

$$\begin{aligned} P_w^l &= P^l + SF_\Theta^l \\ \ell_C^l(P_w^l, Q^l) &= \sum_{p_w^l \in P_w^l} \min_{q^l \in Q^l} \left\| p_w^l - q^l \right\|_2^2 + \sum_{q^l \in Q^l} \min_{p_w^l \in P_w^l} \left\| p_w^l - q^l \right\|_2^2 \end{aligned} \tag{3.8}$$

Smoothness Constraint. In order to enforce local spatial smoothness, we add a smoothness constraint ℓ_S^l, which assumes that the predicted scene flow $SF_\Theta^l(p_j^l)$ in a local region $N(p_i^l)$ of p_i^l should be similar to the scene flow at p_i^l:

$$\ell_S^l(SF^l) = \sum_{p_i^l \in P^l} \frac{1}{|N(p_i^l)|} \sum_{p_j^l \in N(p_i^l)} \left\| SF^l(p_j^l) - SF^l(p_i^l) \right\|_2^2 \tag{3.9}$$

where $|N(p_i^l)|$ is the number of points in the local region $N(p_i^l)$.

Laplacian Regularization. Because the points in a point cloud are only on the surface of a object, their Laplacian coordinate vector approximates the local shape characteristics of the surface, including the normal direction and the mean curvature [121]. The Laplacian coordinate vector $\delta^l(p_i^l)$ can be computed as:

$$\delta^l(p_i^l) = \frac{1}{|N(p_i^l)|} \sum_{p_j^l \in N(p_i^l)} (p_j^l - p_i^l) \tag{3.10}$$

For scene flow, the warped point cloud P_w^l should have the same Laplacian coordinate vector with the second point cloud Q^l at the same position. Since there is no correspondences between the points in P_w^l and Q^l, we firstly compute the Laplacian coordinates $\delta^l(p_i^l)$ for each point in second point cloud Q^l. Then, we interpolate the Laplacian coordinate of Q^l to obtain the Laplacian coordinate on each point p_w^l. We use an inverse distance-based interpolation method similar to Eq.(3.7) to interpolate the Laplacian coordinate δ^l. Let $\delta^l(p_w^l)$ be the Laplacian coordinate of point p_w^l at level l, $\delta^l(q_{inter}^l)$ be the interpolated Laplacian coordinate from Q^l at the same position as p_w^l.

The Laplacian regularization ℓ_L^l is defined as:

$$\ell_L^l(\delta^l(p_w^l), \delta^l(q_{inter}^l)) = \sum_{p_w^l \in P_w^l} \left\| \delta^l(p_w^l) - \delta^l(q_{inter}^l) \right\|_2^2 \tag{3.11}$$

The overall loss is a weighted sum of all losses across all pyramid levels as:

$$\boldsymbol{\ell}(\Theta) = \sum_{l=l_0}^{L} \alpha_l (\beta_1 \ell_C^l + \beta_2 \ell_S^l + \beta_3 \ell_L^l) \tag{3.12}$$

Where α_l is the factor for pyramid level l, $\beta_1, \beta_2, \beta_3$ are the scale factors for each loss respectively. With the self-supervised loss, our model is able to learn the scene flow from 3D point cloud pairs without any ground truth supervision.

3.5 Experiments

In this section, we train and evaluate our PointPWC-Net on the FlyingThings3D dataset [91] with the supervised loss and the self-supervised loss, respectively. Then, we evaluate the generalization ability of our model by first applying the model on the real-world KITTI Scene Flow dataset [97, 95] *without any fine-tuning*. Then, with the proposed self-supervised losses, we further fine-tune our pre-trained model on KITTI dataset to study the best performance we could obtain without supervision. Besides, we also compare the runtime of our model with previous work. Finally, we conduct ablation studies to analyze the contribution of each part of the model and the loss function.

Implementation Details. We build a 4-level feature pyramid from the input point cloud. The weights α are set to be $\alpha_0 = 0.02$, $\alpha_1 = 0.04$, $\alpha_2 = 0.08$, and $\alpha_3 = 0.16$, with weight decay 0.0001.The scale factor β in self-supervised learning are set to be $\beta_1 = 1.0$, $\beta_2 = 1.0$, and $\beta_3 = 0.3$. We train our model starting from a learning rate of 0.001 and reducing by half every 80 epochs. All the hyperparameters are set using the validation set of FlyingThings3D with 8,192 points in each input point cloud.

Evaluation Metrics. For fair comparison, we adopt the evaluation metrics that are used in [42]. Let SF_Θ denote the predicted scene flow, and SF_{GT} be the ground truth scene flow. The evaluate metrics are computed as follows:

Table 3.1: Evaluation results on FlyingThings3D and KITTI dataset. *Self* means self-supervised, *Full* means fully-supervised. All approaches are (at least) trained on FlyingThings3D. On KITTI, *Self* and *Full* refer to the respective models trained on FlyingThings3D that is directly evaluated on KITTI, while *Self+Self* means the model is firstly trained on FlyingThings3D with self-supervision, then fine-tuned on KITTI with self-supervision as well. *Full+Self* means the model is trained with full supervision on FlyingThings3D, then fine-tuned on KITTI with self-supervision. ICP [9], FGR [180], and CPD [99] are traditional method that does not require training. Our model outperforms all baselines by a large margin on all metrics

Dataset	Method	Sup.	EPE3D(m)↓	Acc3DS↑	Acc3DR↑	Outliers3D↓	EPE2D(*px*)↓	Acc2D↑
Flyingthings3D	ICP(rigid) [9]	*Self*	0.4062	0.1614	0.3038	0.8796	23.2280	0.2913
	FGR(rigid) [180]	*Self*	0.4016	0.1291	0.3461	0.8755	28.5165	0.3037
	CPD(non-rigid) [99]	*Self*	0.4887	0.0538	0.1694	0.9063	26.2015	0.0966
	PointPWC-Net	*Self*	**0.1213**	**0.3239**	**0.6742**	**0.6878**	**6.5493**	**0.4756**
	FlowNet3D [78]	*Full*	0.1136	0.4125	0.7706	0.6016	5.9740	0.5692
	SPLATFlowNet [123]	*Full*	0.1205	0.4197	0.7180	0.6187	6.9759	0.5512
	original BCL [42]	*Full*	0.1111	0.4279	0.7551	0.6054	6.3027	0.5669
	HPLFlowNet [42]	*Full*	0.0804	0.6144	0.8555	0.4287	4.6723	0.6764
	PointPWC-Net	*Full*	**0.0588**	**0.7379**	**0.9276**	**0.3424**	**3.2390**	**0.7994**
KITTI	ICP(rigid) [9]	*Self*	0.5181	0.0669	0.1667	0.8712	27.6752	0.1056
	FGR(rigid) [180]	*Self*	0.4835	0.1331	0.2851	0.7761	18.7464	0.2876
	CPD(non-rigid) [99]	*Self*	0.4144	0.2058	0.4001	0.7146	27.0583	0.1980
	PointPWC-Net(w/o ft)	*Self*	*0.2549*	*0.2379*	*0.4957*	*0.6863*	*8.9439*	*0.3299*
	PointPWC-Net(w/ ft)	*Self + Self*	**0.0461**	**0.7951**	**0.9538**	**0.2275**	**2.0417**	**0.8645**
	FlowNet3D [78]	*Full*	0.1767	0.3738	0.6677	0.5271	7.2141	0.5093
	SPLATFlowNet [123]	*Full*	0.1988	0.2174	0.5391	0.6575	8.2306	0.4189
	original BCL [42]	*Full*	0.1729	0.2516	0.6011	0.6215	7.3476	0.4411
	HPLFlowNet [42]	*Full*	0.1169	0.4783	0.7776	0.4103	4.8055	0.5938
	PointPWC-Net(w/o ft)	*Full*	*0.0694*	*0.7281*	*0.8884*	*0.2648*	*3.0062*	*0.7673*
	PointPWC-Net(w/ ft)	*Full + Self*	**0.0430**	**0.8175**	**0.9680**	**0.2072**	**1.9022**	**0.8669**

- *EPE3D(m)*: $\|SF_\Theta - SF_{GT}\|_2$ averaged over each point in meters.
- *Acc3DS*: the percentage of points with *EPE3D* $< 0.05m$ or relative error $< 5\%$.
- *Acc3DR*: the percentage of points with *EPE3D* $< 0.1m$ or relative error $< 10\%$.
- *Outliers3D*: the percentage of points with *EPE3D*$> 0.3m$ or relative error $> 10\%$.
- *EPE2D(px)*: 2D end point error obtained by projecting point clouds back to the image plane.
- *Acc2D*: the percentage of points whose *EPE2D* $< 3px$ or relative error $< 5\%$.

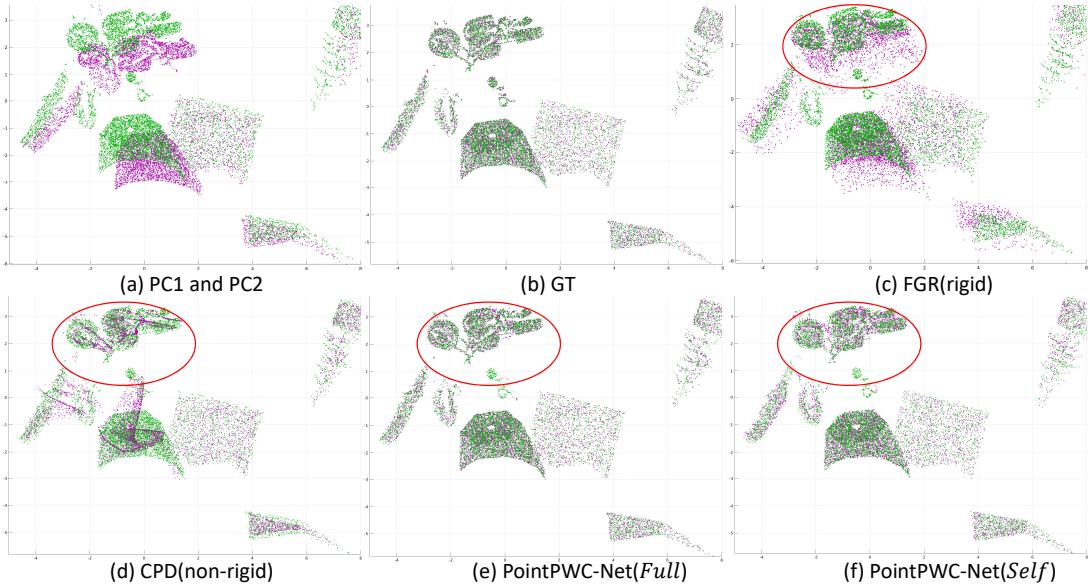


Figure 3.5: Results on FlyingThings3D dataset. In (a), 2 point clouds PC1 and PC2 are presented in Magenta and Green, respectively. In (b-f), PC1 is warped to PC2 based on the (computed) scene flow. (b) shows the ground truth; (c) Results from FGR(rigid) [180]; (d) Results from CPD(non-rigid) [99]; (e) Results from PointPWC-Net(*Full*); (f) Results from PointPWC-Net(*Self*). Red ellipses indicate locations with significant non-rigid motion. Enlarge images for better view. (Best viewed in color)

3.5.1 Supervised Learning

First we conduct experiments with supervised loss. To our knowledge, there is no publicly available large-scale real-world dataset that has scene flow ground truth from point clouds (The input to the KITTI scene flow benchmark is 2D), thus we train our PointPWC-Net on the synthetic Flyingthings3D dataset, following [42]. Then, the pre-trained model is directly evaluated on KITTI Scene Flow 2015 dataset without any fine-tuning.

Train and Evaluate on FlyingThings3D. The FlyingThings3D training dataset includes 19,640 pairs of point clouds, and the evaluation dataset includes 3,824 pairs of point clouds. Our model takes $n = 8,192$ points in each point cloud. We first train the model with $\frac{1}{4}$ of the training set(4,910 pairs), and then fine-tune it on the whole training set, to speed up training.

Table 3.1 shows the quantitative evaluation results on the Flyingthings3D dataset. Our method outperforms all the methods on all metrics by a large margin. Compared to FlowNet3D, our cost volume layer is able to capture the motion information better. Comparing to SPLATFlowNet, original BCL, and HPLFlowNet, our method avoids the preprocessing step of building a permutohedral lattice from the input. Besides, our method outperforms HPLFlowNet on *EPE3D* by 26.9%. And, we are the only method with *EPE2D* under 4px, which improves over HPLFlowNet by 30.7%. See Fig.3.5(e) for example results.

Evaluate on KITTI w/o Fine-tune. To study the generalization ability of our PointPWC-Net, we directly take the model trained using FlyingThings3D and evaluate it on KITTI Scene Flow 2015 [95, 97] *without any fine-tuning*. KITTI Scene Flow 2015 consists of 200 training scenes and 200 test scenes. To evaluate our PointPWC-Net, we use ground truth labels and trace raw point clouds associated with the frames, following [78, 42]. Since no point clouds and ground truth are provided on test set, we evaluate on all 142 scenes in the training set with available point clouds. We remove ground points with height $< 0.3m$ following [42] for fair comparison with previous methods.

From Table 3.1, our PointPWC-Net outperforms all the state-of-the-art methods, which demonstrates the generalization ability of our model. For *EPE3D*, our model is the only one below $10cm$, which improves over HPLFlowNet by 40.6%. For *Acc3DS*, our method outperforms both FlowNet3D and HPLFlowNet by 35.4% and 25.0% respectively. See Fig.3.6(e) for example results.

3.5.2 Self-supervised Learning

Acquiring or annotating dense scene flow from real-world 3D point clouds is very expensive, so it would be interesting to evaluate the performance of our self-supervised approach. We train our model using the same procedure as in supervised learning, i.e. first train the model with one quarter of the training dataset, then fine-tune with the whole training set. Table 3.1 gives the quantitative results on PointPWC-Net with self-supervised learning. We compare our method with ICP(rigid) [9], FGR(rigid) [180] and CPD(non-rigid) [99]. Because traditional point registration methods are not trained with ground truth, we can view them as self/un-supervised methods.

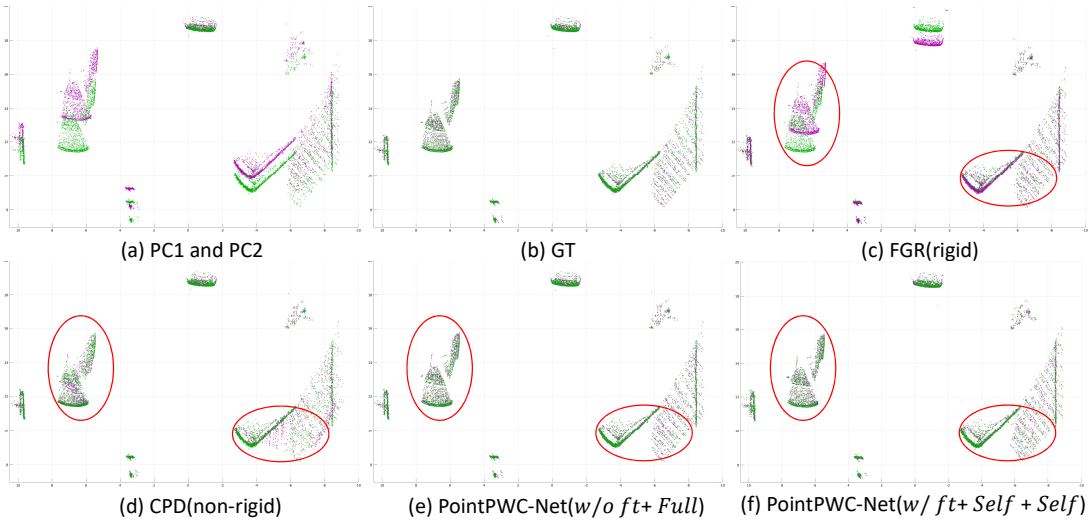


Figure 3.6: Results on KITTI Scene Flow 2015 dataset. In (a), 2 point clouds PC1 and PC2 are presented in Magenta and Green, respectively. In (b-f), PC1 is warped to PC2 based on the (computed) scene flow. (b) shows the ground truth; (c) Results from FGR(rigid) [180]; (d) Results from CPD(non-rigid) [99]; (e) Results from PointPWC-Net(*w/o ft+Full*) that is trained with supervision on FlyingThings3D, and directly evaluate on KITTI without any fine-tune; (f) Results from PointPWC-Net(*w/ ft + Self + Self*) which is trained on FlyingThings3D and fine-tuned on KITTI using the proposed self-supervised loss. Red ellipses indicate locations with significant non-rigid motion. Enlarge images for better view. (Best viewed in color)

Train and Evaluate on FlyingThings3D. We can see that our PointPWC-Net outperforms the traditional methods on all the metrics with a large margin. See Fig.3.5(f) for example results.

Evaluate on KITTI w/o Fine-tuning. Even only trained on FlyingThings3D without ground truth labels, our method can obtain *0.2549m* on *EPE3D* on KITTI, which improves over CPD(non-rigid) by *38.5%*, FGR(rigid) by *47.3%*, and ICP(rigid) by *50.8%*.

Fine-tune on KITTI. With proposed self-supervised loss, we are able to fine-tune the FlyingThings3D trained models on KITTI without using any ground truth. In Table 3.1, the row *PointPWC-Net(w/ ft) Full+Self* and *PointPWC-Net(w/ ft) Self+Self* show the results. *Full+Self* means the model is trained with supervision on FlyingThings3D, then

Table 3.2: Model design. A learnable cost volume preforms much better than traditional cost volume used in PWC-Net [127]. Using our cost volume instead of the MLP+Maxpool used in FlowNet3D's flow embedding layer improves performance by 20.6%. Compared to no warping, the warping layer improves the performance by 40.2%

Component	Status	EPE3D(m)↓
Cost Volume	PWC-Net [127]	0.0821
	MLP+Maxpool(learnable) [78]	0.0741
	Ours(learnable)	**0.0588**
Warping Layer	w/o	0.0984
	w	**0.0588**

fine-tuned on KITTI without supervision. *Self+Self* means the model is firstly trained on FlyingThings3D, then fine-tuned on KITTI both using self-supervised loss. With KITTI fine-tuning, our PointPWC-Net can achieve $EPE3D < 5cm$. Especially, our *PointPWC-Net(w/ ft) Self+Self*, which is fully trained without any ground truth information, achieves similar performance on KITTI as the one that utilized FlyingThings3D ground truth. See Fig.3.6(f) for example results.

3.5.3 Ablation Study and Visualization

Ablation Study. We further conduct ablation studies on model design choices and the self-supervised loss function. On model design, we evaluate the different choices of cost volume layer and removing the warping layer. On the loss function, we investigate removing the smoothness constraint and Laplacian regularization in the self-supervised learning loss. All models in the ablation studies are trained using FlyingThings3D, and tested on the FlyingThings3D evaluation dataset.

Tables 3.2, 3.3 show the results of the ablation studies. In Table 3.2 we can see that our design of the cost volume obtains significantly better results than the cost volume in PWC-Net [127] and FlowNet3D [78], and the warping layer is crucial for performance. In Table 3.3, we see that both the smoothness constraint and Laplacian regularization improve the performance in self-supervised learning. In Table 3.4, we report the runtime of our PointPWC-Net, which is comparable with other deep learning based methods and

Table 3.3: Loss functions. The Chamfer loss is not enough to estimate a good scene flow. With the smoothness constraint, the scene flow result improves by 38.2%. Laplacian regularization also improves slightly

Chamfer	Smoothness	Laplacian	EPE3D(m)↓
✓	-	-	0.2112
✓	✓	-	0.1304
✓	✓	✓	**0.1213**

Table 3.4: Runtime. Average runtime(ms) on Flyingthings3D. The runtime for FlowNet3D and HPLFlowNet is reported from [42] on a single Titan V. The runtime for our PointPWC-Net is reported on a single 1080Ti

Method	Runtime(ms)↓
FlowNet3D [78]	130.8
HPLFlowNet [42]	98.4
PointPWC-Net	117.4

much faster than traditional ones.

Visualization. Fig.3.7 provide more visualization on KITTI dataset of scene flow results by PointPWC-Net. In Fig.3.7, the model is trained with self-supervised loss on FlyingThings3D [91], and directly tested on KITTI Scene Flow 2015 [97, 95] without any finetune. From Fig.3.7, we can see that our model can recover scene flow not only for the rigid objects, such as cars, but also for the non-rigid objects, such as bushes, trees, etc.

Typical Error Types. We summaries three typical error types of our PointPWC-Net for KITTI dataset. To visualize errors, we use blue, green and red to represent the first point cloud, the warped points which are correctly predicted, and the wrongly predicted points, respectively, as shown in Fig. 3.8. The first error type is when the object is a straight line or a plane. In this case, it is hard for the network to construct a cost volume with strong discernment, as shown in Fig.3.8 **A** and **C**. The second one is that it is hard to find good correspondences between consecutive frames due to the strong deformation of local shapes, as shown in Fig.3.8 **B**. The third case is that the ground points are not removed properly, as shown in Fig.3.8 **D**. By using a better ground removal strategy, we can further improve our results.

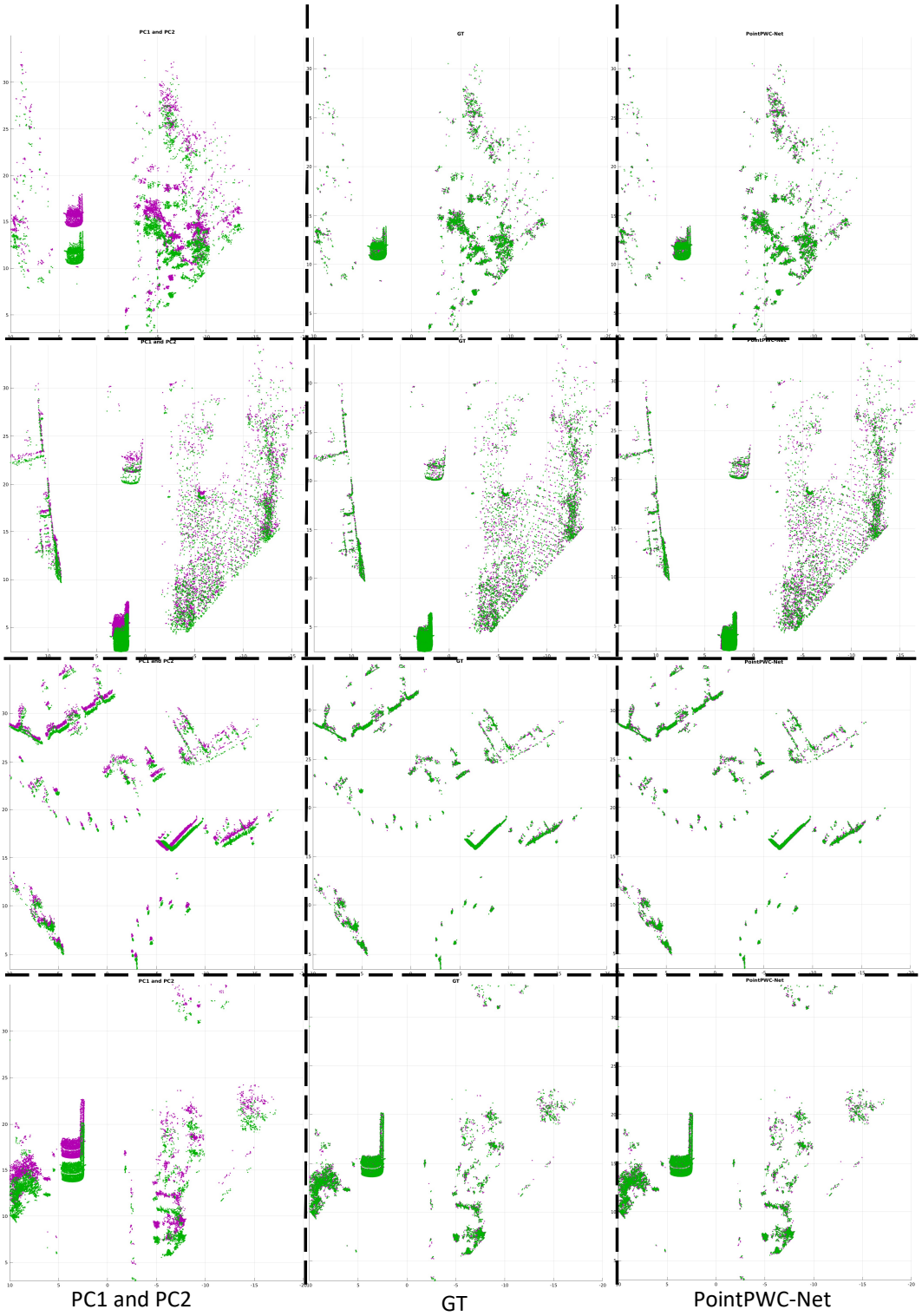


Figure 3.7: Scene Flow Results on KITTI Scene Flow 2015. In (a), 2 point clouds PC1 and PC2 are presented in Magenta and Green, respectively. In (b) and (c), PC1 is warped to PC2 based on the (computed) scene flow. (b) shows the ground truth; (c) Results from PointPWC-Net that is trained with supervision on FlyingThings3D, and directly evaluate on KITTI without any fine-tune. Enlarge images for better view. (Best viewed in color)

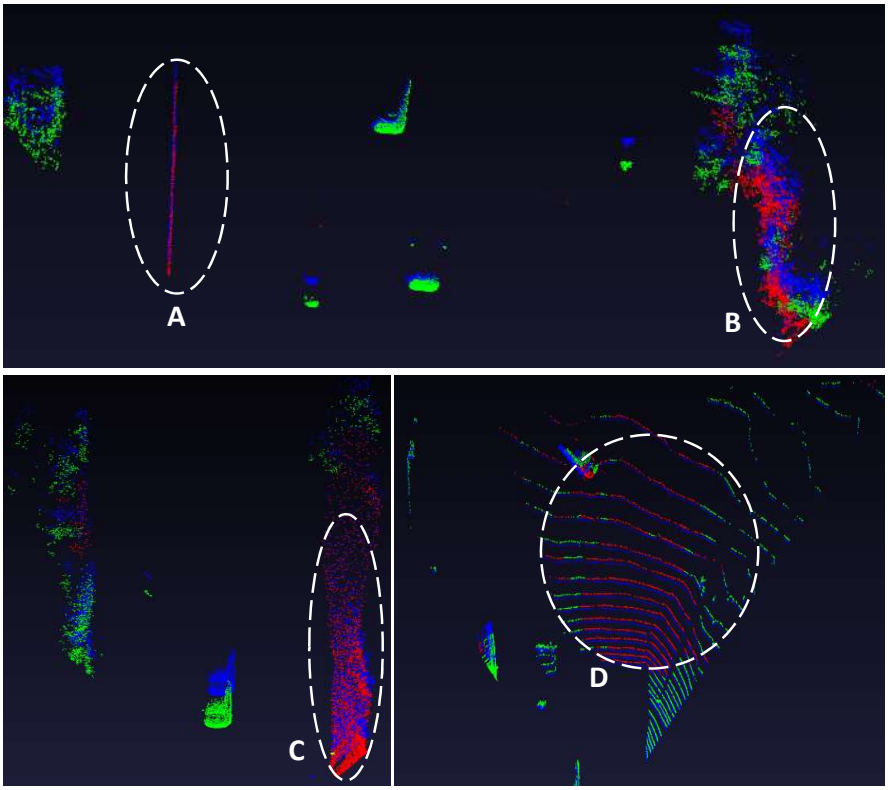


Figure 3.8: Typical error types for KITTI. The blue points are from the first point cloud P. The green points are the warped points $P_w = P + SF$ according to the correctly predicted flow. The "correctness" is measured by Acc3DR. The red points are wrongly predicted. A and C are the ambiguity in 3D point clouds, which are straight lines or plane walls. B is the messy bushes, whose features do not have strong correspondences. D is the case when the ground points are not removed cleanly.

3.6 Conclusion

To better estimate scene flow directly from 3D point clouds, we proposed a novel learnable cost volume layer along with some auxiliary layers to build a coarse-to-fine deep network, called PointPWC-Net. Because of the fact that real-world ground truth scene flow is hard to acquire, we introduce a loss function that train the PointPWC-Net without supervision. Experiments on the FlyingThings3D and KITTI datasets demonstrates the effectiveness of our PointPWC-Net and the self-supervised loss function, obtaining state-of-the-art results that outperform prior work by a large margin.

Chapter 4: PointConvFormer

4.1 Introduction

Sensors for indoor and outdoor 3D scanning have significantly improved in the last decade, in terms of both performance and affordability. Hence, their common output data format, 3D point clouds, has drawn significant attention from academia and industry in recent years. Understanding the 3D real world from 3D point clouds can be applied to many application domains, such as robotics, autonomous driving, CAD, and AR/VR. However, unlike image pixels arranged in regular grids, 3D points are unstructured which makes applying classic grid based Convolutional Neural Networks (CNNs) very difficult.

Various approaches have been proposed in response to this challenge. [124, 71, 15, 60, 67] introduce interesting ways to project a 3D point cloud back to 2D image space to apply 2D convolution. These approaches relies heavily on the choice of projection planes, and have challenges to handle occlusions in the 3D space [177]. Another line of research directly voxelizes 3D space and apply 3D discrete convolution. These methods induce massive computation and memory overhead [89, 120]. Sparse convolution operations [39, 20] partially relieve the limitations, however, doesn't adapt well to the sampling density of the point cloud, reducing their efficiency. Some approaches directly operate on point clouds without discretizing into grids [106, 108, 123, 136, 158, 75]. [106, 108] are pioneers which aggregate information on point clouds using max-pooling layers. Others proposed reordering the input points with a learned transformation [76], a flexible point kernel [136], and a convolutional operation that directly work on point clouds [151, 158] by utilizing a multi-layer perceptron (MLP) to learn the convolution weights implicitly as a nonlinear transformation from the relative positions of the local neighbourhood. Despite these efforts, state-of-the-art often employs a fusion between voxel and point-based methods, complicating the model and reducing efficiency.

Recently, researchers have introduced visual transformers to 3D point clouds process-

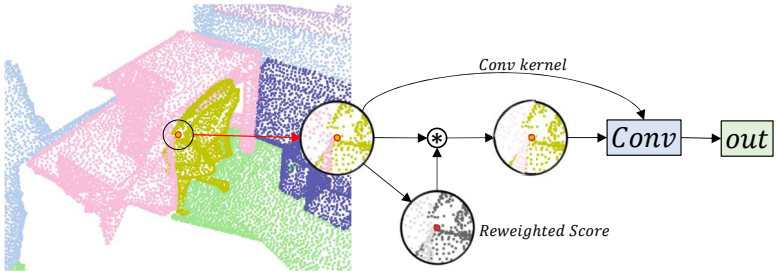


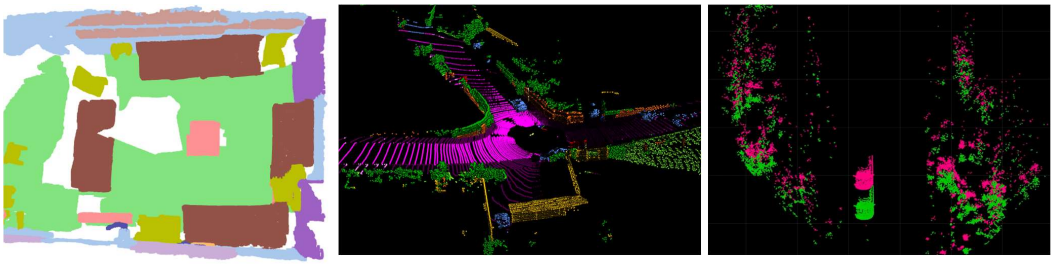
Figure 4.1: PointConvFormer can be seen as a point convolution, but modulated by an attention weight for each point in the neighborhood, computed from the differences of current layer features between neighboring points

ing, inspired by the success in NLP and image analysis [177, 102]. The self-attention mechanism has also been adapated for point clouds processing [177]. However, so far, transformers have not shown to significantly outperform convolutional approaches on point clouds. Both convolution and attention aim to conduct feature aggregation in neighborhoods with high feature correlations. The advantage of convolution is that its translation-invariance usually offers good generalization power. However, attention weights can help locate points that are more correlated with each other.

In this work, we propose *PointConvFormer*, an operation that uses attention weights to modulate a convolution operation, essentially selecting relevant points to perform convolution, with the hope to get the best of both worlds. We also experiment with the multi-head mechanism commonly used in transformers.We evaluate PointConvFormer on two point cloud tasks, semantic segmentation and scene flow estimation. For semantic segmentation, experiment results on the indoor ScanNet [23] and the outdoor SemanticKitti [7] demonstrate superior performances over classic convolution and transformers with a more compact network. We also apply PointConvFormer as the backbone of PointPWC-Net [159] for scene flow estimation, and observe significant improvements on FlyingThings3D [90] and KITTI scene flow 2015 [96] datasets. We include ablation studies which explores the design space of PointConvFormer.

To summarize the main contributions:

- We introduce PointConvFormer which modifies convolution by an attention weight computed from the differences of local neighbourhood features. We further extend



(a)Indoor Scene Segmantation (b)Outdoor Scene Segmantation (c)Scene Flow from Point Clouds

Figure 4.2: Applications of PointConvFormer. PointConvFormer can serve as the backbone for various 3D scene understanding tasks, such as semantic segmentation for indoor/outdoor scenes, and scene flow estimation from point clouds

the PointConvFormer with a multi-head mechanism.

- We conduct thorough experiments on semantic segmentation tasks for both indoor and outdoor scenes, as well as scene flow estimation from 3D point clouds on multiple datasets. Extensive ablation studies are conducted to study the properties and design choice of PointConvFormer.

4.2 Related Work

We examine related work on voxel-based networks, point-based networks, and recent point transformer work for 3D scene understanding. We also review dynamic filtering for 2D images.

Voxel-based networks. Different from 2D images, 3D point clouds are unordered and scattered in 3D space. One of the trending approaches to process 3D point clouds is to voxelize the point clouds into regular 3D voxels. However, directly applying 3D convolution [89, 120] onto the 3D voxels can incur massive computation and memory overhead, which limits its applications to large-scale real world scenarios. [114] propose to use unbalanced octrees with hierarchical partitions. The sparse convolution [39, 20] reduces the convolutional overhead by only working on the non-empty voxels. However, this kind of approaches may still suffer from losing geometric details due to quantization on the voxel grid. The best performances are achieved with high quantization resolutions (e.g. $2cm$ per voxel), which still have high memory consumption.

Point-based networks. There are plenty of work [106, 108, 158, 75, 123, 152] focusing on directly processing point clouds without re-projection or voxelization. [106, 108] propose to use MLPs followed by max-pooling layers to encode and aggregate point cloud features. However, max-pooling could lose critical geometric information in the point cloud. A number of work [92, 59, 91, 72, 38, 147] build a kNN graph from the point cloud and conduct message passing using graph convolution. Later on, [151, 163, 136, 158, 88, 76, 31, 75] conduct continuous convolution on point clouds. [151] represents the convolutional weights with MLPs. SpiderCNN [163] uses a family of polynomial functions to approximate the convolution kernels. [123] projects the whole point cloud into a high-dimensional grid for rasterized convolution. [158, 136] formulate the convolutional weights to be a function of relative position in a local neighbourhood, where the weights can be constructed according to input point clouds. [75] improves over [158] by introducing hand-crafted viewpoint-invariant coordinate transforms to increase the robustness of the network.

Dynamic filters and Transformer. Recently, the design of dynamic convolutional filters [166, 175, 16, 59, 149, 122, 172, 153, 137, 86, 58, 179] has drawn more attentions. This line of work [86, 175, 16, 166] introduces different methods to predict convolutional filters, which are shared across the whole input. [59, 172, 153, 137] propose to predict the complete convolutional filters for each pixel. However, their applications are constrained by their computational inefficiency and high memory usage. [179] introduces decoupled dynamic filters with respect to the input features on 2D classification and upsampling tasks. [122, 129] propose to re-weight 2D convolutional kernels with a fixed Gaussian or Gaussian mixture model for pixel-adaptive convolution. Dynamic filtering share some similarities with the popular transformers, whose weights are functions of feature correlations. However, the dynamic filters are mainly designed for images instead of point clouds.

With recent success in natural language processing [27, 24, 142, 157, 168] and 2D images analysis [48, 29, 176, 110], transformers have drawn more attention in the field of 3D scene understanding. Some work [68, 79, 167, 161] utilize global attention on the whole point cloud. However, these approaches introduces heavy computation overhead and are unable to extend to large scale real world scenes, which usually contain over $100k$ points per point cloud scan. Recently, the work [177, 102] introduce point transformer

with local attention to reduce the computation overhead, which could be applied to large scenes. Compared to previous convolutional approaches, our PointConvFormer computes the weights with both the relative position and the feature difference. Compared to transformers, the attention of the PointConvFormer modulates convolution kernels and use the sigmoid activation instead of softmax. Experiments showed that our design significantly improves the performance of the networks.

4.3 PointConvFormer

4.3.1 Review of Point Convolutions and Transformers

Given an input continuous signal $x(p) \in \mathbb{R}^{c_{in}}$ where $p \in \mathbb{R}^s$ with s being a small number (usually 2 for 2D images or 3 for 3D point clouds, but could be any arbitrary low-dimensional Euclidean space), $x(\cdot)$ can be sampled as a point cloud $P = \{p_1, \ldots, p_n\}$ with the corresponding values $x_P = \{x(p_1), \ldots, x(p_n)\}$, where each $p_i \in \mathbb{R}^s$. The continuous convolution at point p is formulated as:

$$Conv(w, x)_p = \int_{\Delta p} \langle w(\Delta p), x(p + \Delta p) \rangle d\Delta p \tag{4.1}$$

where $w(\Delta p) \in \mathbb{R}^{c_{in}}$ is the continuous convolution weight function. Inspired by the continuous formulation of convolution, [158, 152] discretize the continuous convolution on a neighbourhood of point p. Let $X_{p_i} \in \mathbb{R}^{c_{in}}$ be the input feature of p_i the discretized convolution on point clouds is written as:

$$X'_p = \sum_{p_i \in \mathcal{N}(p)} w(p_i - p)^\top X_{p_i} \tag{4.2}$$

where $\mathcal{N}(p)$ is a neighborhood that is normally chosen as the k-nearest neighbor or ϵ-ball neighborhood of the center point p. The function $w(p_i - p) : \mathbb{R}^s \mapsto \mathbb{R}^{c_{in}}$ can be approximated as an MLP.

In PointConv [158], an efficient formulation was derived when $w(p_i - p)$ has a linear final layer $w(p_i - p) = W_l h(p_i - p)$, where $h(p_i - p) : \mathbb{R}^3 \mapsto \mathbb{R}^{c_{mid}}$ is the output of the penultimate layer of the MLP and $W_l \in \mathbb{R}^{c_{in} \times c_{mid}}$ is the learnable parameters in the

final linear layer. We can equivalently change Eq. (4.2) on the neighbourhood $\mathcal{N}(p)$ into,

$$X'_p = \left\langle \text{vec}(W_l), \text{vec}\left\{ \sum_{p_i \in \mathcal{N}(p)} h(p_i - p) X_{p_i}^\top \right\} \right\rangle . \tag{4.3}$$

where $\text{vec}(\cdot)$ turns the matrix into a vector. Note that W_l represents parameters of a linear layer and hence independent of p_i. Thus, when there are c_{out} convolution kernels, n training examples with a neighborhood size of k each, there is no longer a need to store the original convolution weights $w(p_i - p)$ for each point in each neighborhood with a dimensionality of $c_{out} \times c_{in} \times k \times n$. Instead, the dimension of all the $h(p_i - p)$ vectors in this case is only $c_{mid} \times k \times n$, where c_{mid} is significantly smaller(usually 8 or 16) than $c_{out} \times c_{in}$ (could go higher than $10^2 \times 10^2$). This efficient PointConv enables applications to large-scale networks on 3D point cloud processing.

Recently, transformer architectures are popular with 2D images. 3D point cloud-based transformers have also been proposed (e.g. [177, 102]). Transformers compute an attention model between points (or pixels) based on the features of both points and the positional encoding of them. Relative positional encoding was the most popular which encodes $w(p_i - p)$, similar to Eq. (4.2). It has been shown to outperform absolute positional encodings in many papers [116, 21, 177]. Adopting similar notations to Eq. (4.2), we can express the softmax attention model used in transformers as:

$$\mathit{Attention}(p) = \sum_{p_i \in \mathcal{N}(p)} \text{softmax}(\mathbf{q}(X_{p_i})\mathbf{k}(X_p) + w(p_i - p)) \cdot \mathbf{v}(X_{p_i}) \tag{4.4}$$

where $\mathbf{q}(\cdot), \mathbf{k}(\cdot), \mathbf{v}(\cdot)$ are transformation to the features to form the query, key and value matrices respectively, usually implemented with MLPs. One can see that there are similarities and differences between PointConv [158] and the attention model [142]. First, both employ $w(p_i - p)$, but in PointConv that is the sole source of the convolutional kernel which is translation-invariant. In attention models, the matching between the query transform $\mathbf{q}(X_{p_i})$ and the key transform $\mathbf{k}(X_p)$ of the features are also considered, which is no longer translation-invariant.

Another important difference to note is that in attention models the final attention value is an output from the softmax function. Note that softmax output has a range of $[0, 1]$

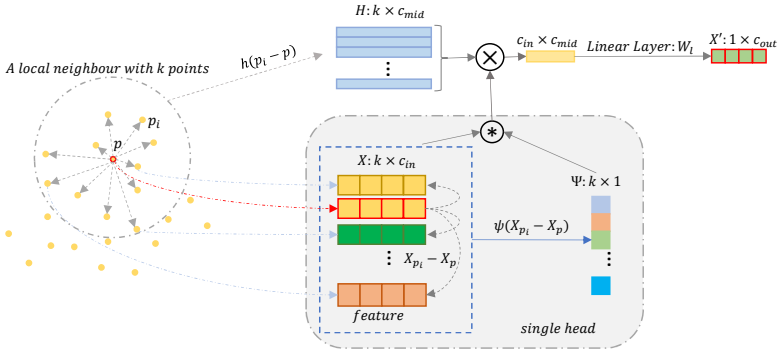


Figure 4.3: Details of the PointConvFormer Operation. $h(p_i - p) : \mathbb{R}^3 \mapsto \mathbb{R}^{k \times c_{mid}}$ and $\psi(X_{p_i} - X_p) : \mathbb{R}^{c_{in} \mapsto \mathbb{R}}$ are functions of the relative position $p_i - p$ and functions of differences of features. The weights of PointConvFormer combines the information from feature differences $X_{p_i} - X_p$ and relative position $p_i - p$.

which is limited to **non-negative** weights at each point, which means the output of eq. (4.4) is a non-negative weighted average of the features of the input. To us, it is a bit curious why this is the right idea, as we tend to believe each neighborhood point could have positive and negative impacts to the features of the center point, and limiting it only to non-negative could be a specific design choice that needs further investigation.

We want to note that the $\mathbf{v}(\cdot)$ transform can be seen as a 1×1 convolution on the input features. It is common to insert 1×1 convolution layers between regular convolution layers in deep architectures (e.g. ResNet[44]), hence we could view it as an additional 1×1 convolution layer before the attention layer, hence we can compare the attention layer and PointConv without considering $\mathbf{v}(\cdot)$.

4.3.2 PointConvFormer Layer

We are interested to adopt the strengths of attention-based models, while exploring the design space where we still preserve some of the benefits of convolution and explore the possibility of having negative weights. Inspired by the discussion above, We define a convolution operation with its weights as functions of both the relative position $p_i - p$ and the feature difference $X_{p_i} - X_p$. Hence, PointConvFormer layer of a point p with its

neighbourhood $\mathcal{N}(p)$ can be written as:

$$X'_p = \sum_{p_i \in \mathcal{N}(p)} w(p_i - p)^\top \psi(X_{p_i} - X_p) X_{p_i} \tag{4.5}$$

where the function $w(p_i - p)$ is the same as defined in Eq. (4.2), the function $\psi(X_{p_i} - X_p) : \mathbb{R}^{c_{in}} \mapsto \mathbb{R}$ is the function of feature differences $X_{p_i} - X_p$, which projects the differences of features in a local neighbourhood to a re-weighted attention $\Psi_{X_{p_i} - X_p} \in \mathbb{R}$, similar to the attention scores in transformers [177].

If we fix the function $\psi(X_{p_i} - X_p) = 1$, the PointConvFormer layer is equivalent to Eq. (4.3), which reduces to traditional convolution. $\psi(X_{p_i} - X_p)$ is approximated with another MLP followed by a regularization layer, such as softmax, sigmoid, or ReLU. We explore the optimal choice of the MLP structure in ablation studies. As a result, the function $w(p_i - p)$ learns the weights respect to the relative positions, and the function $\psi(X_{p_i} - X_p)$ learns the differences between the features of point p and its neighbourhood, which works similarly to the attention in transformer. However, different from the transformer whose non-negative weights are directly used as a weighted average on the input, the output of $\psi(X_{p_i} - X_p)$ modifies the convolutional filter $w(p_i - p)$, which allows each neighborhood point to have both positive and negative contributions.

Since $\Psi_{X_{p_i} - X_p}$ is a re-weighted score that works on the input feature, we adopt the same approach in PointConv [158] to create an efficient version of the PointConvFormer layer. Following eq.(4.3), we have:

$$X'_p = W_l \sum_{p_i \in \mathcal{N}(p)} h(p_i - p) \psi(X_{p_i} - X_p) X_{p_i}^\top \tag{4.6}$$

where W_l and $h(\cdot)$ are the same as in Eq. (4.3). The PointConvFormer structure is illustrated in Fig. 4.3.

Some theoretical intuitions about why PointConvFormer may work can be drawn from the Gaussian complexity theory of CNNs. We note the following bound proved in [74]:

$$\hat{G}_N(F) \leq C \max_{p' \in \mathcal{N}(p)} \sqrt{\mathbb{E}_{X,p}[(X_p - X_{p'})^2]} \tag{4.7}$$

where $\hat{G}_N(F)$ is the empirical Gaussian complexity on the function class F: a one-layer CNN followed by a fully-connected layer, and C is a constant. A *smaller* Gaussian complexity leads to better generalization [4]. To minimize Gaussian complexity, we should select points that has high feature correlation to belong to the same neighborhood of the network. Conventional CNNs achieve better generalization by choosing nearby points (e.g. 3×3) as neighborhoods which are naturally more correlated than faraway points [74]. In PointConvFormer, $\psi(X_{p_i} - X_p)$ especially with a sigmoid activation function, allows the network to dynamically choose to activate or deactivate neighbor point p_i based on its feature difference with the center point p, which can in turn further lower the Gaussian complexity of the trained network and improve its generalization power.

4.3.3 Multi-Head Mechanism

As in Eq.(4.6), the weight function $\psi(X_{p_i} - X_p) : \mathbb{R}^{c_{in}} \mapsto \mathbb{R}$ learns the relationship between the center point feature $X_p \in \mathbb{R}^{c_{in}}$ and its neighbourhood features $X_{p_i} \in \mathbb{R}^{c_{in}}$, where c_{in} is the number of the input feature dimension. To increase the representation power of the PointConvFormer, we attempt to use the multi-head mechanism to learn different kind of relationships. As a result, the function $\psi : \mathbb{R}^{c_{in}} \mapsto \mathbb{R}$ becomes a set of functions $\psi_i : \mathbb{R}^{c_{in}} \mapsto \mathbb{R}$ with the number of heads being h, and $i \in \{1, ..., h\}$. Different heads in the multi-head mechanism will correspondence to different channels of the input feature, as in [142]. Hence, the PointConvFormer with multi-head could encode multiple relationships in the feature space, which improve the representation capabilities.

4.4 Semantic Segmentation

To demonstrate the effectiveness of the proposed PointConvFormer in real world point clouds, we adopt the PointConvFormer to large-scale semantic segmentation tasks. In this section, we introduce the network structure and the main network components used for the segmentation.

4.4.1 PointConvFormer Block

To build deep neural network for various computer vision tasks, we construct bottleneck residual blocks with PointConvFormer layer as its main components. The detailed structures of the residual blocks are illustrated in Fig. 4.4. The input of the residual block is the input point features $X \in \mathbb{R}^{c_{in}}$ along with its coordinates $p \in \mathbb{R}^3$. The residual block uses a bottleneck structure, which consists of two branches. The residual branch is a linear layer followed by PointConvFormer layer followed by another linear layer. The shortcut branch can be formulated in three different ways depending on the output feature size. If the output feature has the same cardinality and dimensionality, the shortcut branch is just a identity mapping. If the output feature has the same cardinality but with different dimensionality, the shortcut branch is a linear mapping. If the output feature has different cardinality, e.g. when the point cloud is downsampled, the shortcut branch can use max-pooling layers to aggregate features.

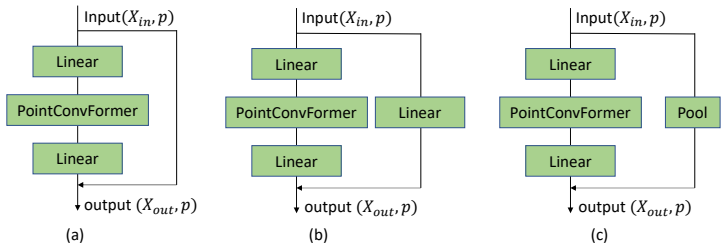


Figure 4.4: The residual blocks of PointConvFormer. We use Linear layers and pooling layers to change the dimensionality and cardinality of the shortcut to match the output of the residual branch.

4.4.2 Backbone Structure

In this work, we adopted a general U-Net structure with residual blocks in the encoding layers as our backbone model, where the point clouds are gradually downsampled to coarse resolution, then gradually upsampled to its original resolution with the help of finer level features. Please refer to the supplementary for detailed network structure. We use the grid-subsampling method [136] to downsample the point clouds as in [136] along with the PointConvFormer blocks to encode features. For upsampling layers, we are unable to apply PointConvFormer because for points that do not exist in the

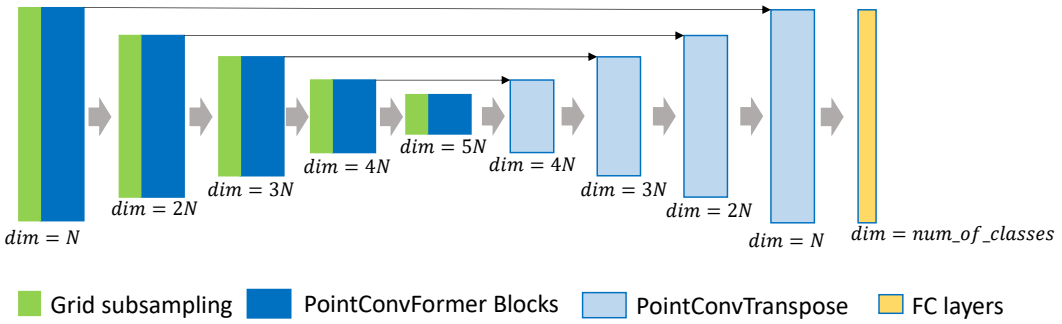


Figure 4.5: The network structure of semantic segmentation. We use a U-Net structure for semantic segmentation tasks. The U-Net contains 5 resolution levels. For each resolution level, we use grid subsampling to downsample the input point clouds, then followed by several pointconvformer residual blocks. For deconvolution, we just use PointConv as described in the main paper. We set $N = 64$ for ScanNet [23] Dataset and $N = 48$ for SemanticKitti [6] Dataset. (Best viewed in color.)

downsampled cloud, their features are not present. To address this issue, we note that in eq. (4.3) of PointConv, p itself does not have to belong to $\mathcal{N}(p)$, thus we can just apply PointConv layers for deconvolution without features X_p as long as coordinates p are known. As in Fig. 4.5, the U-Net we adopt contains 5 resolution levels. For each resolution level, we use grid subsampling to downsample the input point clouds, then followed by several pointconvformer residual blocks.

4.5 Scene Flow estimation from Point Clouds

In this section, we adopt our PointConvFormer to scene flow estimation from 3D point clouds. Scene flow is the 3D displacement vector between each surface in two consecutive frames. As a fundamental tool for low-level understanding of the world, scene flow can be used in many 3D applications, such as autonomous driving, virtual reality, etc. Traditionally, scene flow was estimated directly from RGB/RGBD data [53, 93, 146]. However, with the recent development of 3D sensors such as LiDAR and 3D deep learning techniques, there is increasing interest on directly estimating scene flow from 3D point clouds [78, 42, 159, 105, 156]. In this work, we are interested in adopting the PointConvFormer into the PointPWC-Net [159], which utilizes a coarse-to-fine framework for

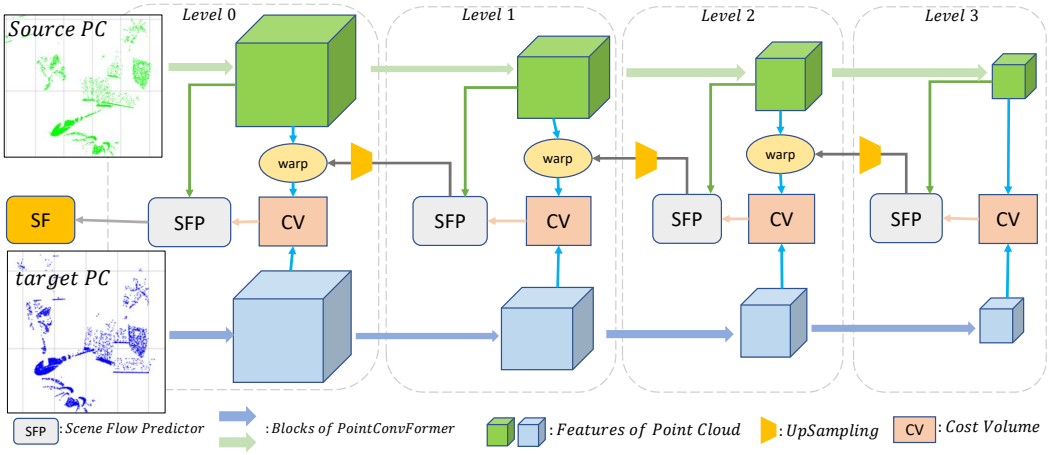


Figure 4.6: The network structure of PointPWC-Net with PointConvFormer. The feature pyramid is built with blocks of PointConvFormers. As a result, there are 4 resolution levels in the PointPWC-Net. At each level, the features of the source point cloud are warped according to the upsampled coarse flow. Then, the cost volume are computed using the warped source features and target features. Finally, the scene flow predictor predicts finer flow at the current level using a PointConv with features from the first point cloud, the cost volume, and the upsampled flow. (Best viewed in color.)

scene flow estimation.

PointPWC-Net [159] is a coarse-to-fine network design, which aims to iteratively refine the scene flow estimation. It mainly contains 5 modules, including the feature pyramid network, cost volume layers, upsampling layers, warping layers, and the scene flow predictors. The feature pyramid network is built with multiple PointConv [158] layers to encode point features in different resolutions. The cost volume layers compute the learned cost volume in the 3D space. The upsampling layers and warping layers interpolate the coarse scene flow to a finer level and warp the target point cloud, respectively. Finally, the scene flow predictors estimate scene flow for each resolution. To adopt our PointConvFormer to the PointPWC-Net, we replace the PointConv in the Feature pyramid layers with the PointConvFormer and keep the rest of the structure the same as the original version of PointPWC-Net for fair comparison. Fig. (4.6) illustrates the overview of the PointPWC-Net with our PointConvFormer. Please note that for fair comparison, we keep the input and output feature dimentionality exactly the same as the original

PointPWC-Net.

4.6 Experiments

In this section, we conduct experiments in a number of domains and tasks to demonstrate the effectiveness of the proposed PointConvFormer. For 3D semantic Segmentation, we use the challenging ScanNet [23], which is a large-scale indoor scene dataset, and the SemanticKitti dataset [7], which is a large-scale outdoor scene dataset. Besides, we conduct experiments on the scene flow estimation from 3D point clouds with the synthetic FlyingThings3D dataset [90] for training and KITTI scene flow 2015 dataset [96] for testing. We also conduct ablation studies to explore the properties of the PointConvFormer.

Implementation Details. We implement PointConvFormer in PyTorch [103]. We use the Adam optimizer with $(0.9, 0.999)$ betas and 0.0001 weight decay. For the ScanNet dataset, we train the model with an initial learning rate 0.001 and dropped to 0.5x for every 80 epochs for 400 epochs. For the SemanticKitti dataset, the model is trained with an intial learning rate 0.001 and dropped to 0.5x for every 8 epochs for 40 epochs. Both semantic segmentation tasks are trained with weighted cross entropy loss. For the scene flow estimation, we follow the exact same training pipeline in [159] for fair comparison.

4.6.1 Indoor Scene Semantic Segmentation

We conduct 3D semantic scene segmentation on the ScanNet [23] dataset. We use the official split with $1,201$ scenes for training and 312 for validation. MinkowskiNet42 [20], SparseConvNet [39] are compared as representative voxel-based methods. The PointNet [106], PointConv [158], VI-PointConv [75], PointASNL [164], and KPConv [136] are chosen as representative point-based methods. Recently, there are work adopting transformer to point clouds. We chose the Point Transformer [177] and the fast point transformer [102] as representative transformer based methods. Since the Point Transformer does not report their results on the ScanNet dataset, we adopt their point transformer layer (a standard multi-head attention layer) with the same network structure as ours. Hence, it serves as a direct comparison between PointConvFormer and multi-head at-

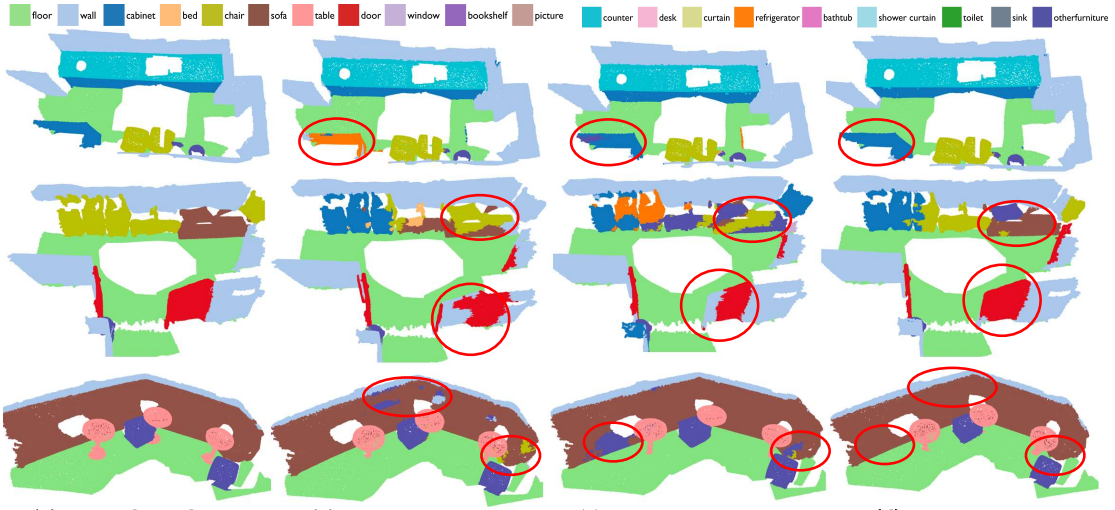
tention. There exists some other approaches [18, 50, 51, 66] which use additional inputs, such as 2D images, which benefit from ImageNet [26] pre-training that we do not use. Hence, we excluded these methods from comparison, accordingly.

We evaluate the models on the ScanNet validation split. From the results shown in Table 4.1, we can see that our proposed PointConvFormer achieves the best performance among the point-based methods, outperforming the best convolution based method KPConv [136] by 3.6% in mIoU. Besides, our PointConvFormer achieves slightly better results than the voxel-based methods, including the MinkowskiNet42, with only 40% of the number of learnable parameters of MinkowskiNet42. In the result visualizations shown in Fig. 4.7, we observe that PointConvFormer is able to achieve better predictions with fine details comparing with PointConv [158] and Point Transformer [177]. Interestingly, it seems that PointConvFormer is usually able to find the better prediction out of PointConv [158] and Point Transformer [177], showing that its novel design brings the best out of both operations.

Following [102], we further conduct experiments on different input voxel sizes. Since we use grid-subsampling [136] with different grid sizes to downsample the input point cloud, voxel size is also a parameter in our network. Although, we still utilize kNN neighborhoods which always have k neighbors whereas sparse convolution could have far fewer points in their neighborhood, hencing needing more layers than us to get the same receptive field. From the results reported in Table. 4.2, our PointConvFormer outperforms MinkowskiNet42 [20] and Fast Point Transformer [102] on both $10cm$ and $5cm$ setups by a large margin.

4.6.2 Outdoor Scene Semantic Segmentation

The SemanticKitti [7, 36] dataset is a large-scale street view point clouds dataset built upon the KITTI Vision Odometry Benchmark [36]. The dataset is collected in Germany with Velodyne-HDLE64 LiDAR, and consists of $43,552$ point cloud scans sampled from 22 sequences in driving scenes. Each point cloud scan contains $10-13k$ points. We follow the training and validation split in [7] and 19 classes are used for training and evaluation. , and there are $4,071$ scans in sequence 08)for validation. For each 3D point, only the (x, y, z) coordinates are given without any color information. It is a challenging



(a) *Ground Truth* (b) *PointConv* (c) *Point Transformer* (d) *PointConvFormer*

Figure 4.7: ScanNet result visualization. We visualize the ScanNet prediction results from our PointConvFormer, PointConv [158] and Point Transformer [177]. The red ellipses indicates the improvements of our PointConvFormer over other approaches. Points with ignore labels are filtered for a better visualization. (Best viewed in color)

dataset because the scanning density is uneven as faraway points are sparser in LIDAR scans.

Table 4.3 reports the results on the semanticKitti dataset. Because this work mainly focus on the basic building block, PointConvFormer which is applicable to any kind of 3D point cloud data, of deep neural network, we do not compare with work [182, 17] whose main novelties work mostly on LiDAR datasets. We use a simple U-Net structure for semantic segmentation as described in Sec. 4.4.2 which has less parameters than most other high-performing networks. From the table, one can see that our PointConvFormer outperforms both point-based methods and point+voxel fusion methods. Especially, our method obtains better results comparing with SPVNAS [131], which utilizes the network architecture searching (NAS) techniques and fuses both point and voxel branches. We did not utilize any NAS in our system which would only further improve our performance.

Table 4.1: Semantic segmentation results on ScanNet dataset. We use the ScanNet [23] validation set. *For Point Transformer [177], we implemented it on the same network structure as PointConvFormer, hence it also serves as an ablation comparing regular self-attention layers with PointConvFormer layers

Methods	# Params(M)	Input	mIoU(%)
PointNet [106]	-	Point	53.5
PointConv [158]	-	Point	61.7
KPConv *deform* [136]	14.9	Point	69.2
PointASNL [164]	-	Point	63.5
VI-PointConv [75]	15.5	Point	68.2
SparseConvNet [39]	-	Voxel	69.3
MinkowskiNet42 [20]	37.9	Voxel	72.2
Fast Point Transformer [102]	37.9	Voxel	72.0
Point Transformer*	10.7	Point	68.0
PointConvFormer(ours)	15.1	Point	**72.8**

4.6.3 Scene Flow Estimation from Point Clouds

Besides the semantic segmantion tasks, we also conduct experiments on scene flow estimation directly from 3D point clouds using the PointPWC-Net with PointConvFormer, as introduced in Sec. 4.5. For simplicity, we name the new network *'PCFPWC-Net'* where PCF stands for PointConvFormer. To train the PCFPWC-Net, we use the multi-scale supervised loss [159] and follow the training pipeline in [159]. For a fair comparison, we use the same dataset configurations as in [159]. The model is firstly trained on FlyingThings3D [91], which is a large synthetic image dataset for scene flow estimation. The 3D point clouds are reconstructed from image pairs with the depth map provided in the dataset following [42]. As a result, the training dataset contains 19,640 pairs of point clouds, and the evaluation dataset contains 3,824 pairs of point clouds. We adopt the same hyper-parameters used in [159]. There are 4 pyramid levels in PCFPWC-Net. The model is trained with a starting learning rate of 0.001 and dropped by half every 80 epochs. After training on FlyingThings3D, we directly evaluate the trained model on the real world KITTI Scene Flow dataset [94, 96] to test the generalization capabilities of our model. We follow the same preprocessing step in [42] and obtain 142 valid scenes for evaluation. For comparison, we use the same metrics as [159], which uses

Table 4.2: Comparison with different input voxel size. We compare the results on the ScanNet [23] validation set with different input voxel size. † means the results are reported in [102]. We use grid subsampling [136] to downsample the input point clouds, which is similar to voxelization. However, we still use kNN neighborhood after downsampling which is different from the voxel neighborhood used in other approaches.

Methods	Voxel/grid size	# Params(M)	Input	mIoU(%)
MinkowskiNet42† [20]	10cm	37.9	Voxel	60.4
Fast Point Transformer [102]	10cm	37.9	Voxel	65.3
PointConvFormer(ours)	10cm	15.1	Point	**68.0**
MinkowskiNet42† [20]	5cm	37.9	Voxel	66.6
Fast Point Transformer [102]	5cm	37.9	Voxel	70.1
Point Transformer*	5cm	10.7	Point	68.0
PointConvFormer(ours)	5cm	15.1	Point	**72.8**

Table 4.3: Semantic segmentation results on SemanticKitti validation set.

Method	#MACs(G)	# Param.(M)	Input	mIoU(%)
RandLA-Net [49]	66.5	1.2	Point	57.1
FusionNet [173]	-	-	Point+Voxel	63.7
KPRNet [63]	-	-	Point+Range	64.1
MinkowskiNet [20]	113.9	21.7	Voxel	61.1
SPVCNN [131]	118.6	21.8	Point+Voxel	63.8
SPVNAS [131]	64.5	10.8/12.5	Point+Voxel	64.7
PointConvFormer(ours)	91.1	8.1	Point	**65.6**

EPE3D(m) as the main metric, *Acc3DS*, *Acc3DR*, *Outliers3D*, *EPE2D(px)*, *Acc2D* for further comparison. The details are described below.

Evaluation Metrics. For comparison, we use the same metrics as [159]. Let SF_Θ denote the predicted scene flow, and SF_{GT} be the ground truth scene flow. The evaluate metrics are computed as follows:

- *EPE3D(m)*: $\|SF_\Theta - SF_{GT}\|_2$ averaged over each point in meters.
- *Acc3DS*: the percentage of points with $EPE3D < 0.05m$ or relative error $< 5\%$.
- *Acc3DR*: the percentage of points with $EPE3D < 0.1m$ or relative error $< 10\%$.

• *Outliers3D*: the percentage of points with *EPE3D*$> 0.3m$ or relative error $> 10\%$.

• *EPE2D(px)*: 2D end point error obtained by projecting point clouds back to the image plane.

• *Acc2D*: the percentage of points whose *EPE2D* $< 3px$ or relative error $< 5\%$.

Table 4.4: Evaluation results on FlyingThings3D and KITTI dataset. All approaches are trained on FlyingThings3D with the supervised loss. On KITTI, the models are directly evaluated on KITTI without any fine-tuning.

Dataset	Method	EPE3D(m)↓	Acc3DS↑	Acc3DR↑	Outliers3D↓	EPE2D(px)↓	Acc2D↑
Flyingthings3D	FlowNet3D [78]	0.1136	0.4125	0.7706	0.6016	5.9740	0.5692
	SPLATFlowNet [123]	0.1205	0.4197	0.7180	0.6187	6.9759	0.5512
	HPLFlowNet [42]	0.0804	0.6144	0.8555	0.4287	4.6723	0.6764
	HCRF-Flow [73]	0.0488	0.8337	0.9507	0.2614	2.5652	0.8704
	FLOT [105]	0.052	0.732	0.927	0.357	-	-
	PV-RAFT [156]	0.0461	0.8169	0.9574	0.2924	-	-
	PointPWC-Net [159]	0.0588	0.7379	0.9276	0.3424	3.2390	0.7994
	PCFPWC-Net(ours)	**0.0416**	**0.8645**	**0.9658**	**0.2263**	**2.2967**	**0.8871**
KITTI	FlowNet3D [78]	0.1767	0.3738	0.6677	0.5271	7.2141	0.5093
	SPLATFlowNet [123]	0.1988	0.2174	0.5391	0.6575	8.2306	0.4189
	HPLFlowNet [42]	0.1169	0.4783	0.7776	0.4103	4.8055	0.5938
	HCRF-Flow [73]	0.0531	0.8631	0.9444	0.1797	2.0700	0.8656
	FLOT [105]	0.056	0.755	0.908	0.242	-	-
	PV-RAFT [156]	0.0560	0.8226	**0.9372**	0.2163	-	-
	PointPWC-Net [159]	0.0694	0.7281	0.8884	0.2648	3.0062	0.7673
	PCFPWC-Net(ours)	**0.0479**	**0.8659**	0.9332	**0.1731**	**1.7943**	**0.8924**

From Table 4.4, we can see that our proposed PCFPWC-Net outperforms previous methods in almost all the evaluation metrics. Comparing with PointPWC-Net [159], our PCFPWC-Net achieves around 10% improvement in EPE3D and EPE2D on the FlyingThings3D, around 10% in Acc3DS and Acc2D. On the KITTI dataset, our PCFPWC-Net also shows strong result for scene flow estimation by improving the EPE3D by more than 30%($0.0694 \mapsto 0.0479$) over PointPWC-Net [159]). Fig. 4.8 illustrates the qualitative results of PCFPWC-Net for both FlyingThings3D and KITTI dataset.

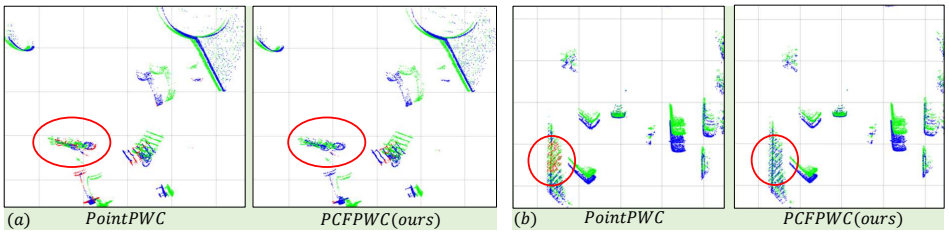


Figure 4.8: Qualitative comparison between PointPWC-Net and PCFPWC-Net. (a) is the visualization of the FlyingThings3D dataset. (b) is the visualization of the KITTI dataset. Green points are the source point cloud. Blue points are the points warped by the correctly predicted scene flow. The predicted scene flow belonging to Acc3DR is regarded as a correct prediction. For the points with incorrect predictions, we use the ground truth scene flow to warp them and the warped results are shown as red points. (Best viewed in color.)

Table 4.5: Ablation Study. We disable each component of the PointConvFormer in turn.

Reweighted score	Conv	mIoU(%)
✓		65.78
	✓	65.20
✓	✓	**69.26**

4.6.4 Ablation Studies

In this section, we perform thorough ablation experiments to investigate our proposed PointConvFormer. The ablation studies are conducted on the ScanNet [23] dataset. For efficiency, we downsample the input point clouds with a grid-subsampling method [136] with a grid size of $10cm$ as in [102].

Effectiveness of different components. We conduct ablation studies by disabling each component of the PointConvFormer in turn. Table. 4.5 reports the experiment results. Without the convolution weights or the reweighted score, the performance drops more than 2% comparing the full PointConvFormer, indicating the effectiveness of our design.

Number of neighbours. We first conduct experiments on the neighbourhood size k

in the PointConvFormer for feature aggregation. The results are reported in Table. 4.6. The best result is achieved with a neighbourhood size of 16. A large neighbourhood size of 32 also obtains comparable result. However, an even larger neighbourhood size of 64 gives worse result, which may cause by introducing excessive less relevant features in a neighbourhood [177].

Table 4.6: Ablation Study. Number of neighbours in each local neighbourhood.

Nieghbourhood Size	4	8	16	32	64
mIoU(%)	63.00	68.10	**69.26**	68.54	67.41

Table 4.7: Ablation Study. Number of heads.

Number of Head	1	2	4	8	16
mIoU(%)	68.32	68.77	68.84	**69.26**	68.72

Number of heads in ψ. As described in Sec. 4.3.3, our PointConvFormer could employ the multi-head mechanism to further improve the representation capabilities of the model. We conduct ablation experiments on the number of heads in the PointConvFormer. The results are shown in Table. 4.7. From Table. 4.7, we find that PointConvFormer already achieves good result with a single head. PointConvFormer with 2 heads slightly improves the segmentation results. More heads in PointConvFormer harm the results.

The structure of MLP in ψ. We conduct experiments to figure out the optimal design of the MLP for the function ψ in the PointConvFormer. The ablations contain two parts: the number of layers in MLP and the last regularization function of MLP. Most of the attention based methods use *softmax* to normalize the attention score. In this experiments, we also test *sigmoid* and *ReLU*. Table. 4.8 and Table. 4.9 report the ablation results. From Table. 4.8, we find that our PointConvFormer achieves reasonable results even with one hidden layer in the MLP of ψ, with the best results obtained with two hidden layers in the MLP. In Table. 4.9, the *sigmoid* activation function obtains the best performance. Interestingly, the performance becomes worse with *softmax*. This might be because of the properties of the softmax function that its attention scores have to be nonnegative and sum to 1, which usually results in only few non-zero outputs.

Table 4.8: Ablation Study. Number of layers in MLP of ψ.

Number of layers	1	2	3	4
mIoU(%)	68.53	**69.26**	68.23	68.16

Table 4.9: Ablation Study. Regularization function.

Regularization	Softmax	ReLU	Sigmoid
mIoU(%)	68.57	68.48	**69.96**

4.7 Visualization

4.7.1 Visualization of Reweighted Scores

Since the reweighted score in PointConvFormer works in local neighbourhoods, it is hard to plot a meaningful visualization of the local neighbourhood in a point cloud. In order to actually see what the reweighted score learn from the dataset, we visualize the difference of the learned reweighted score for some example scenes in the ScanNet [23] dataset. The difference is computed by $score\_max - score\_min$, where $score\_max$ is the maximum reweighted score in the neighbourhood and $score\_min$ is the minimum. A larger difference indicates a strong reweighting in the neighourhood and the PointConvFormer would be more similar to transformer. Otherwise, a smaller difference indicates $\psi = constant$ in the local neighbourhood. With constant or nearly constant reweighted scores throughout the neighbourhood, the PointConvFormer would become a general convolution. We visualize the difference in Fig. 4.9 and Fig. 4.10 with *hot* colormap. From Fig. 4.9 and Fig. 4.10, we can see that higher differences happen mostly in object boundaries. For smooth surfaces and points from the same class, the difference of reweighted scores is low. This visualization further confirms that PointConvFormer is able to utilize feature differences to conduct feature aggregation accordingly.

4.7.2 More Visualization

In this section, we report more visualization of the prediction of our PointConvFormer. Fig. 4.11 is the visualization of the comparison between PointConv [158], Point Transformer [177] and PointConvFormer on the ScanNet dataset [23]. Fig. **??** illustrates the prediction of PointConvFormer on the SemanticKitti dataset [6]. Fig. 4.13 and Fig. 4.14 are the comparison between the prediction of PointPWC [159] and PCFPWC-Net on the FlyingThings3D [90] and the KITTI Scene Flow 2015 dataset [94]. Please also refer

to the video for better visualization.

4.8 PointConvFormer Variant: GuidedConv

In this section, we propose a PointConvFormer variant to improve the performance of predicting dense correspondences on 2D image pairs based on coarse-to-fine framework. In the coarse-to-fine framework, the neighbourhood selection is important especially when upsamples the coarse flow to a finer level. In a local neighbourhood with same or similar flow directions, the flow of the center point of the point clouds could be improved or corrected. A different motion in the neighbourhood could introduce noise to the finer level flow. In the original formulation, the weights of PointConvFormer contains two parts: the weights of the relative position, and the weights of the feature differences. The weights of the feature differences works as a learned filter to reweight the importance of the neighbourhood contribution in the convolution operation. The input of the weights function is the input feature. However, in the coarse-to-fine framework for the dense correspondences, the neighborhood can be defined by different motion directions. The guidance weights learn the relationship of the flow estimation between the neighbourhood points and the center point in local neighbourhoods of a point cloud. With the guidance weights, the convolutions weights is re-weighted according to the correlation of the similarity of the flow in a local region. As a result, the extended PointConvFormer can be written as following:

$$X'_p = W_l \sum_{p_i \in \mathcal{N}(p)} h(p_i - p)\psi(G_{p_i} - G_p)X^{\top}_{p_i} \tag{4.8}$$

where W_l and $h(\cdot)$ are the same as in Eq. (4.6). The G is the guidance feature. In original PointConvFormer, $G = X$, which is the input feature. In the coarse-to-fine framework, we could use the coarse prediction as G to guide the correspondence prediction.

The dense geometric correspondences task [138, 92] is a fundamental problem in computer vision, which is finding pixel-to-pixel correspondences between images that consist of different views of the same scene and include large geometric transformations [138], such as significant scale changes, rotations and shearings, as in Fig. 4.15, which makes it

more difficult than the optical flow task which usually contains continuous motion. In this work, we adopt the PointConvFormer variant to the coarse-to-fine framework [128, 138] to demonstrate the effectiveness of the GuidedConv. We name the variant on 2D images as GuidedConv. As discussed before, the coarse estimation could be used as guidance features G to conduct GuidedConv in finer levels to achieve better refinement.

To demonstrate the effectiveness of GuidedConv, we choose GLU-Net [138] as our baseline model for dense geometric correspondences. GLU-Net[138] is a network structure that is designed to estimate a dense displacement field $\mathbf{w} \in \mathbb{R}^{H\times W\times 2}$ from a pair of images $\mathbf{I}_s \in \mathbb{R}^{H\times W\times 3}$ and $\mathbf{I}_t \in \mathbb{R}^{H\times W\times 3}$. H and W are the spatial size of the image pair. The dense displacement field $\mathbf{w}$ should warp the source image $\mathbf{I}_s$ to target image $\mathbf{I}_t$ so that $\mathbf{I}_t(\mathbf{x}) \approx \mathbf{I}_s(\mathbf{x} + \mathbf{w}(\mathbf{x}))$. Similar to PWC-Net[128], GLU-Net[138] is designed in a coarse-to-fine fashion. The displacement field in the most coarse level is estimated first. Then, the coarse displacement field is gradually refined in a finer level. To capture large-displacements, [138] introduces an adaptive resolution architecture consisting of two subnetworks, which operates on two different image resolutions. The first network **L-Net** downscales the input source and target images to a fixed resolution $H_L \times W_L$ and computes a global cost volume for coarse estimation. The second network **H-Net** directly operates on the original image resolution $H \times W$ for the refinement of the coarse estimation. Please refer to [138] for detailed information.

In order to compare with the traditional convolution, we use the same network structure as in [138]. The main difference between GLU-Net and the network we use is that we conduct 2DGuidedConv with coarse displacement fields as guidance features in the finer level flow estimator, as the Guide SFP in Fig. 4.16. We name the network with GuidedConv as **GuidedGLU-Net**. Fig. 4.16 illustrates the network structure of the GuidedGLU-Net.

4.8.1 Experiment Results

For training, we adopt the multi-scale endpoint error(EPE) loss with respect to the ground truth displacements [138]. For fair comparison, we use the same datasets as [138], which is a combination of the DPED [55], CityScapes [22], and ADE-20K [178] datasets. The total dataset contains 40,000 images with resolutions larger than 750×750. In

training, the image pairs are cropped to 520×520, and synthetic affine transformations are applied to generate ground truth displacement fields [92].

For the task of geometric matching, the images consist of different views of the same scene and include large geometric transformations. We evaluate our GuidedGLU-Net on the HPatches dataset [3] as in [138]. We employ the 59 sequences of the HPatches dataset labeled with v_X. Each image sequence contains a source image and 5 target images taken under increasingly larger viewpoints changes, with sizes ranging from 450×600 to 1613×1210. Similar to [138], we evaluate our GuidedGLU-Net on two different resolutions, the original resolution (noted as **HP**) and the downscaled resolution 240×240 (noted as **HP-240**). We employ the Average EndPoint Error(AEPE) and Percentage of Correct Keypoints(PCK) as our main evaluation metrics. AEPE is the Euclidean distance between estimated and ground truth displacement fields, averaged over all valid pixels of the target image. PCK is the percentage of EPE smaller than a certain threshold δ.

The experiment results on the HPatches dataset are shown in Table 4.10. We use the decoupled dynamic convolution(DDF) [179] in the GLU-Net as the DDF [179]+GLU-Net in Table 4.10. From the table, one can see that our GuidedGLU-Net achieves the best results in all the metrics. Especially, our GuidedGLU-Net outperforms the GLU-Net on PCK-1px of **HP-240** by around 8%, and **HP** by around 3%. Both metrics indicate that our GuidedConv can greatly improve the accuracy of the dense correspondences. We also show the detailed AEPE results on different viewpoint changes in Table 4.11. There are five different viewpoint changes in the HPatches dataset. The viewpoint changes increases from VP I to VP V. VP I has the smallest viewpoint changes and VP V has the largest viewpoint changes. Our GuidedGLU-Net improves the AEPE for all the viewpoint changes in **HP-240** comparing with GLU-Net. In **HP**, our GuidedGLU-Net works much better than GLU-Net on large viewpoint changes(VP IV and VP V). Fig. 4.18 illustrates the qualitative results of GuidedGLU-Net. Besides, we also visualize some of the continuous guidance function ψ in Eq. (4.8) with respect to the difference of displacement fields in Fig. 4.17, which shows that different Ws clearly favor the selection of some coarse displacement values over others.

Ablation studies on different guidance features. We further conduct experiments

Table 4.10: Evaluation on the HPatches datasets. Our GuidedGLU-Net obtains the best performance on all the metrics. Lower AEPE and higher PCK are better.

	HP-240			**HP**		
	AEPE↓	PCK-1px↑	PCK-5px↑	AEPE↓	PCK-1px↑	PCK-5px↑
LiteFlow-Net [54]	19.41	28.36%	57.66%	118.85	13.91%	31.64%
PWC-Net [128]	21.68	20.99%	54.19%	96.14	13.14%	37.14%
DGC-Net [92]	9.07	50.01%	77.40%	33.26	12.00%	58.06%
GLU-Net [138]	7.40	59.92%	83.47%	25.05	39.55%	78.54%
DDF [179]+GLU-Net	7.07	64.40%	85.74%	24.91	39.88%	78.53%
GuidedGLU-Net	**6.95**	**68.40%**	**87.33%**	**24.06**	**42.25%**	**80.41%**

Table 4.11: AEPE results on the HPatches datasets with different viewpoints(VP). The viewpoint changes increase from VP I to VP V, with VP I the smallest viewpoint change and VP V the largest viewpoint change.

Method	**HP-240(AEPE)↓**					
	VP I	VP II	VP III	VP IV	VP V	ALL
GLU-Net [138]	0.59	4.05	7.64	9.82	14.89	7.40
GuidedGLU-Net(Ours)	**0.49**	**3.59**	**7.60**	**8.71**	**14.37**	**6.95**
Method	**HP(AEPE)↓**					
	VP I	VP II	VP III	VP IV	VP V	ALL
GLU-Net [138]	**1.55**	12.66	**27.54**	32.04	51.47	25.05
GuidedGLU-Net(Ours)	1.64	**12.10**	27.98	**30.24**	**48.33**	**24.06**

Table 4.12: Ablation studies on different guidance features. The coarse flow works best as guidance features comparing with color information and pixel features. Lower AEPE and higher PCK are better.

	HP-240			**HP**		
Guidance Features	AEPE↓	PCK-1px↑	PCK-5px↑	AEPE↓	PCK-1px↑	PCK-5px↑
Color	7.35	**69.62%**	86.70%	26.38	41.92%	80.30%
Features	**6.83**	68.38%	87.16%	25.40	41.63%	80.28%
Coarse Flow	6.95	68.40%	**87.33%**	**24.06**	**42.25%**	**80.41%**

Table 4.13: Comparison with the state-of-the-art dynamic filters. Our GuidedGLU-Net obtains the best performance on all the metrics. Lower AEPE and higher PCK are better.

	HP-240			**HP**		
	AEPE↓	PCK-1px↑	PCK-5px↑	AEPE↓	PCK-1px↑	PCK-5px↑
GLU-Net [138]	7.40	59.92%	83.47%	25.05	39.55%	78.54%
PAC [122]+GLU-Net	7.58	66.39%	85.97%	26.37	38.43%	76.77%
DDF [179]+GLU-Net	7.07	64.40%	85.74%	24.91	39.88%	78.53%
GuidedGLU-Net	**6.95**	**68.40%**	**87.33%**	**24.06**	**42.25%**	**80.41%**

using different features as guidance features in GuidedConv to study the better choice of guidance features. The experiment results are shown in Table. 4.12. The features we experiment on are the color information from the source image, the features from the source feature pyramid and the coarse flow that is used in our GuidedGLU-Net. From Table. 4.12, although the color and features perform better on AEPE and PCK-1px of **HP-240**, respectively, the coarse flow works better as guidance features in most of the evaluation metrics.

Compare with state-of-the-art dynamic filters. In order to compare with state-of-the-art dynamic filters [122, 138], we conduct experiments on the dense geometric correspondences on 2D image pairs with different dynamic filters, including the pixel-adaptive convolution(PAC) [122], and decoupled dynamic filter(DDF) [179]. The PAC [122] proposes to re-weight 2D convolutional kernels with a fixed Gaussian kernel for pixel-adaptive convolution. And, the DDF [179] improves the efficiency of dynamic filters by decoupling to dynamic spatial filters and dynamic channel filters for classification and upsampling tasks. Since both methods could be used as a replacement of the traditional convolution, we replace the convolution operation in flow estimation network of GLU-Net with the dynamic filters discussed above, as the GuidedGLU-Net in the main paper. For fair comparison, we use the same training pipeline and hyper-parameter to train the PAC+GLU-Net, DDF+GLU-Net, and GuidedGLU-Net. The results are shown in Table 4.13. From Table 4.13, the GuidedGLU-Net achieves the best performance on all the metrics.

4.9 Conclusion

In this work, we propose a novel point cloud layer, PointConvFormer, which can be widely used in various computer vision tasks. Unlike traditional convolution of which convolutional kernels are functions of the relative position, the convolutional weights of the PointConvFormer are functions of both the relative position and the difference of features. By taking the feature differences into account, the PointConvFormer incorporates benefits of attention models, which could help the network to focus on points with high feature correlation during feature encoding. Thorough experiments on a number of point clouds tasks showed that PointConvFormer significantly outperforms traditional point-based operations and outperforms other voxel-based or point-voxel fusion approaches, with significantly less trainable parameters than voxel-based or fusion approaches. Furthermore, we extend the PointConvFormer to coarse-to-fine matching framework, named GuidedConv. The GuidedConv utilizes coarse predictions as guidance features to filter the convolutional neighbourhood for a better finer level prediction. Experiments on dense geometric correspondences from 2D image pairs demonstrate the effectiveness of the GuidedConv over traditional rasterized and continuous convolutions.

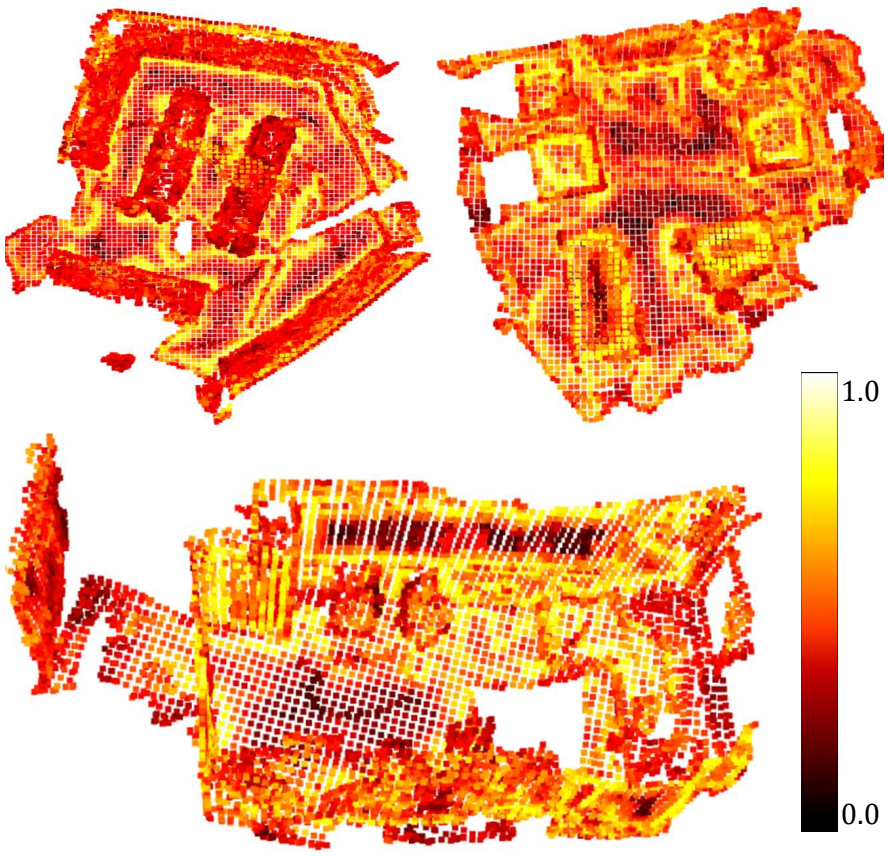


Figure 4.9: Visualization of reweighted scores(part 1). We visualize the difference of the learned reweighted scores in each neighbourhood. The difference is compute by $score\_max - score\_min$, where $score\_max$ is the maximum reweighted score in the neighbourhood and $score\_min$ is the minimum. The yellow(higher difference) indicates the neighbourhood would contain points from different classes. The dark red(low difference) indicates the neighbourhood would contain points from the same class. We visualize point clouds with $10cm$ grid and hot colormap. (Best viewed in color.)

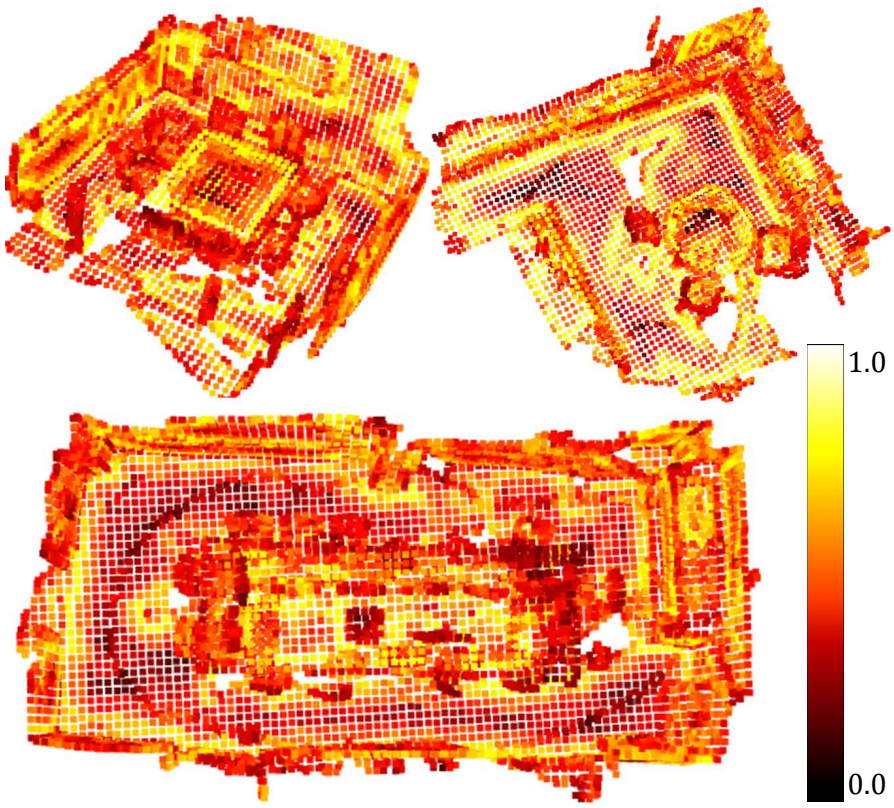


Figure 4.10: Visualization of reweighted scores(part 2). We visualize the difference of the learned reweighted scores in each neighbourhood. The difference is compute by $score\_max - score\_min$, where $score\_max$ is the maximum reweighted score in the neighbourhood and $score\_min$ is the minimum. The brighter color(higher difference) indicates the neighbourhood would contain points from different classes. The darker color(low difference) indicates the neighbourhood would contain points from the same class. We visualize point clouds with $10cm$ grid and hot colormap. (Best viewed in color.)

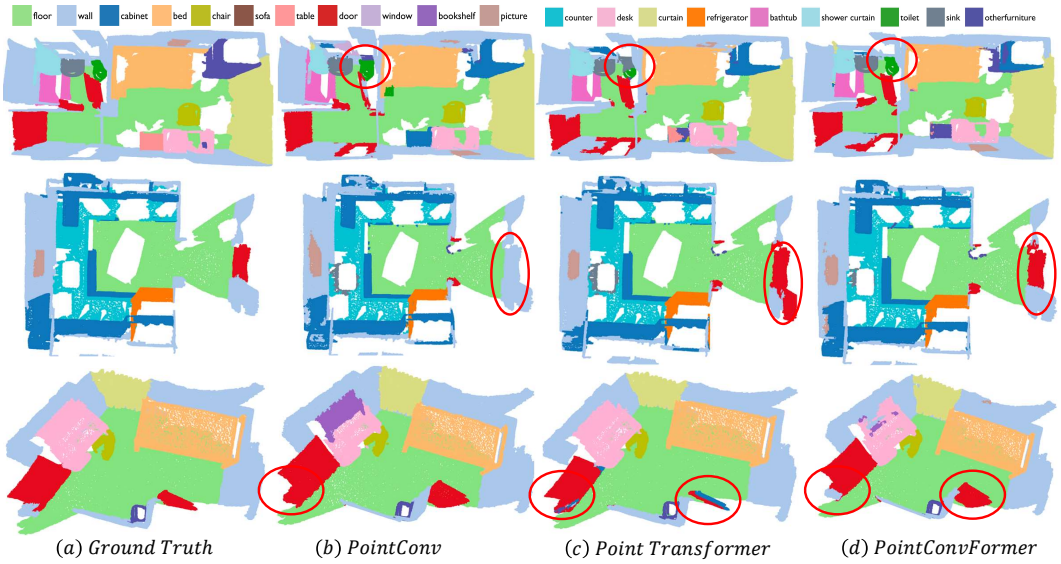


Figure 4.11: ScanNet result visualization. We visualize the ScanNet prediction results from our PointConvFormer, PointConv [158] and Point Transformer [177]. The red ellipses indicates the improvements of our PointConvFormer over other approaches. Points with ignore labels are filtered for a better visualization. (Best viewed in color)

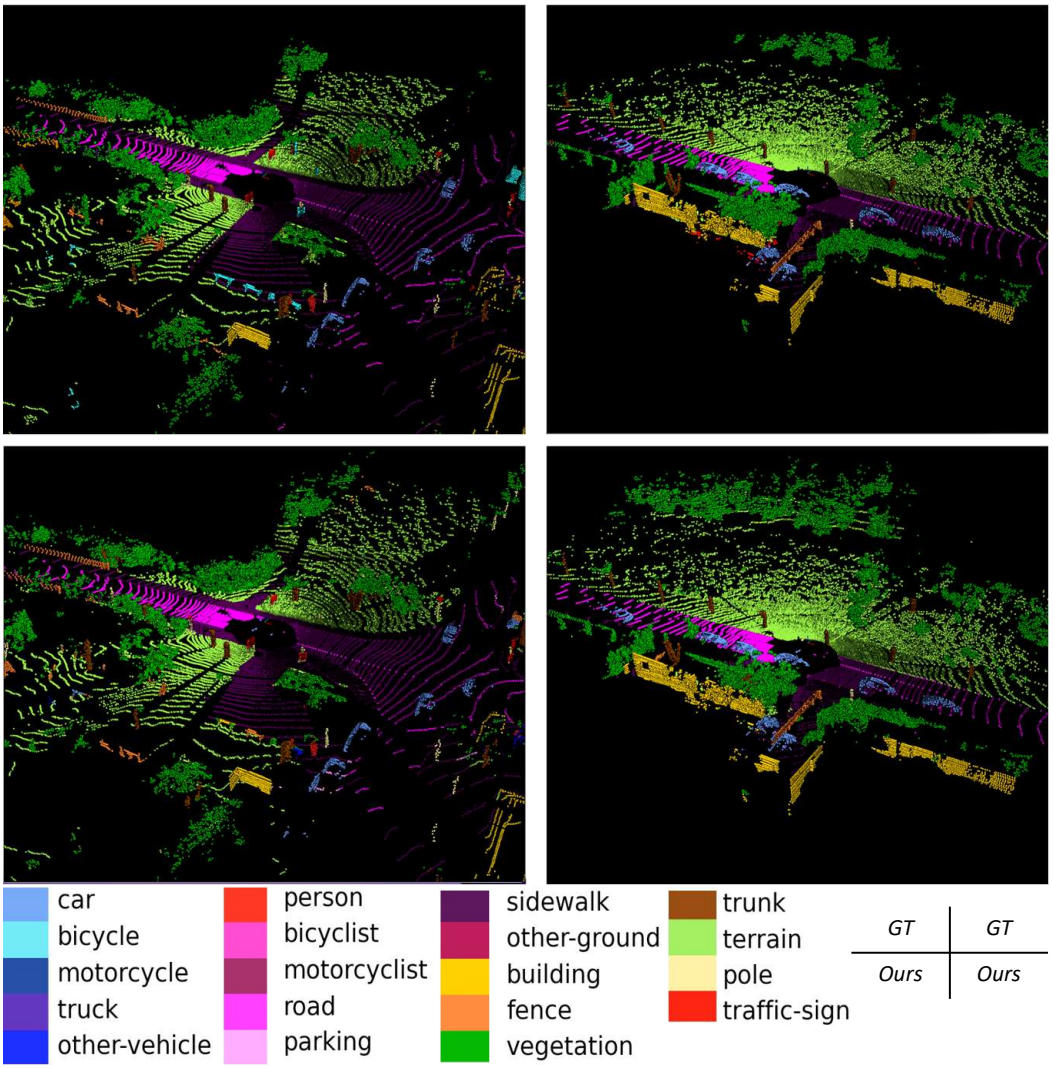


Figure 4.12: SemanticKitti result visualization. We visualize the SemanticKitti prediction results from our PointConvFormer. Each column is a scan from SemanticKitti validation set. The first row is the input, the second row is the ground truth, the third row is our prediction. (Best viewed in color.)

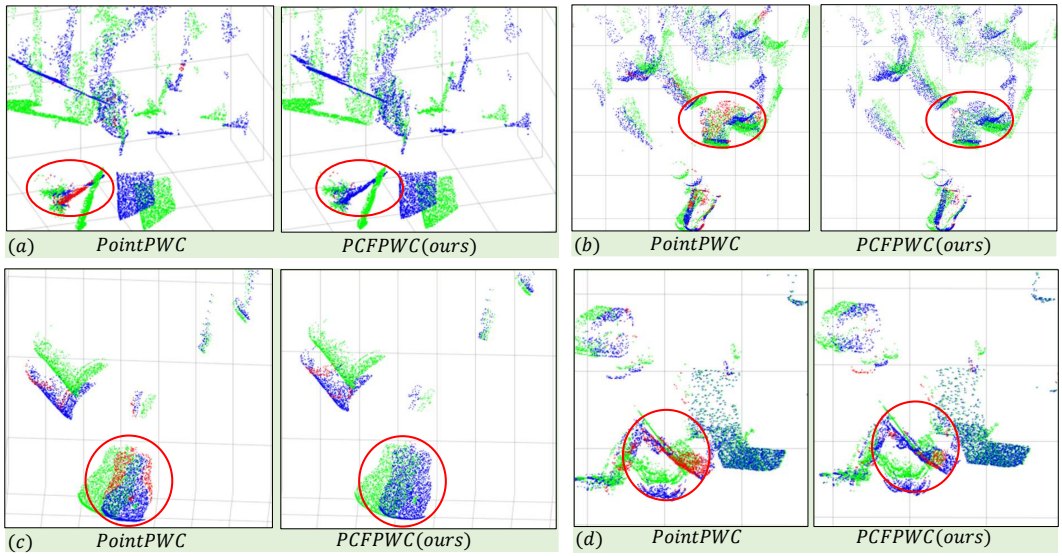


Figure 4.13: Qualitative comparison between PointPWC-Net and PCFPWC-Net (FlyingThings3D [91]). (a) is the visualization of the FlyingThings3D dataset. (b) is the visualization of the KITTI dataset. Green points are the source point cloud. Blue points are the points warped by the correctly predicted scene flow. The predicted scene flow belonging to Acc3DR is regarded as a correct prediction. For the points with incorrect predictions, we use the ground truth scene flow to warp them and the warped results are shown as red points. (Best viewed in color.)

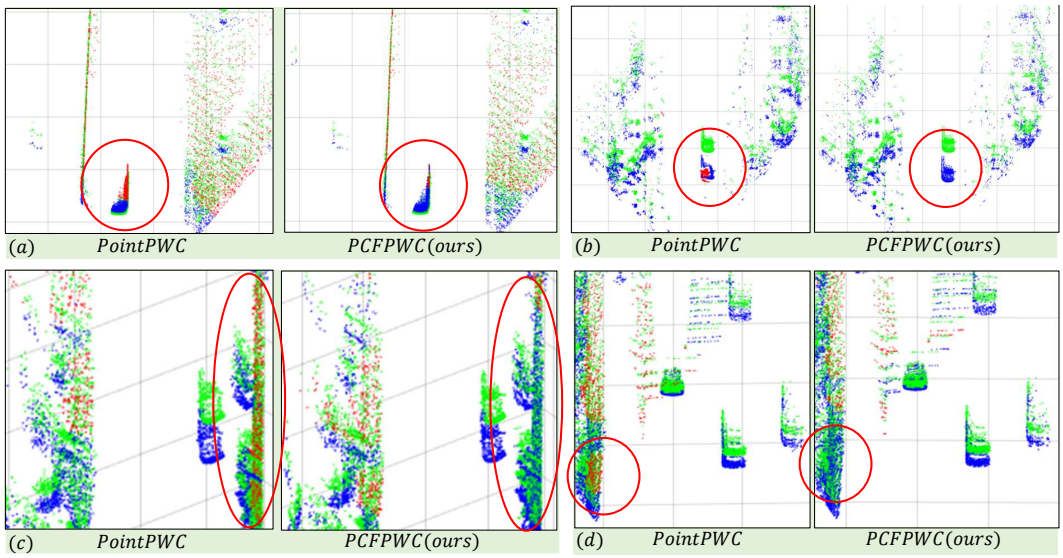


Figure 4.14: Qualitative comparison between PointPWC-Net and PCFPWC-Net (KITTI [94]). Green points are the source point cloud. Blue points are the points warped by the correctly predicted scene flow. The predicted scene flow belonging to Acc3DR is regarded as a correct prediction. For the points with incorrect predictions, we use the ground truth scene flow to warp them and the warped results are shown as red points. (d) is a failure case, where the points on the wall or ground/road are hard to find accurate correspondences for both PointPWC and PCFPWC. (Best viewed in color.)

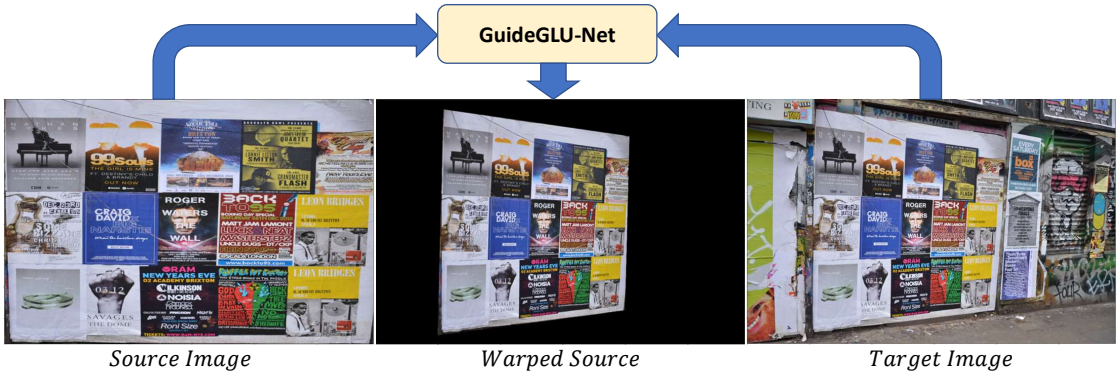


Figure 4.15: Dense geometric correspondences on 2D image pairs. Given a source image and a target image that has undergone significant affine transformations, the task is to find the dense displacement field between the input images.

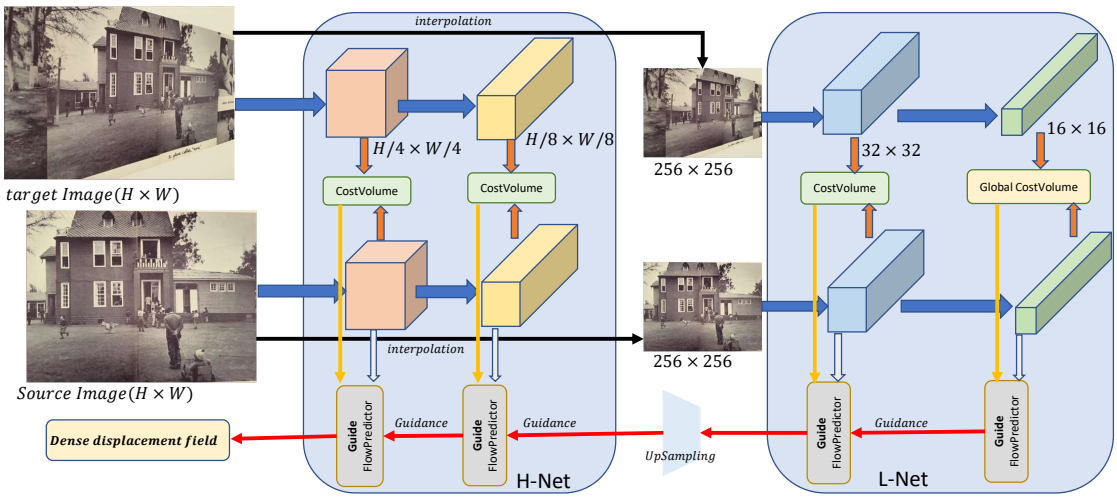


Figure 4.16: The network structure of GuidedGLU-Net. GuidedGLU-Net contains two sub-networks, H-Net and L-Net. L-Net processes fixed resolution(256×256) images, H-Net processes original resolution images. The main differences between GuidedGLU-Net and GLU-Net is that we use the coarse estimation as guidance for finer level estimation. (Best viewed in color)

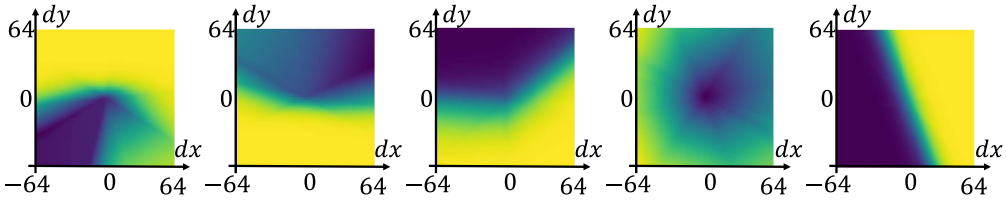


Figure 4.17: Examples of trained Guidance filters W w.r.t the displacement. We plot the pre-trained guidance filters ψ from the GuidedConv in Eq. (**??**). The ranges of the difference of the displacement are chosen from -64 to 64. When the coarse flow difference between center and its neighbor is indicated in dark blue (as opposed to bright yellow), it has lower guidance weights, which means the corresponding convolution kernel weights are made smaller. (Best viewed in color)

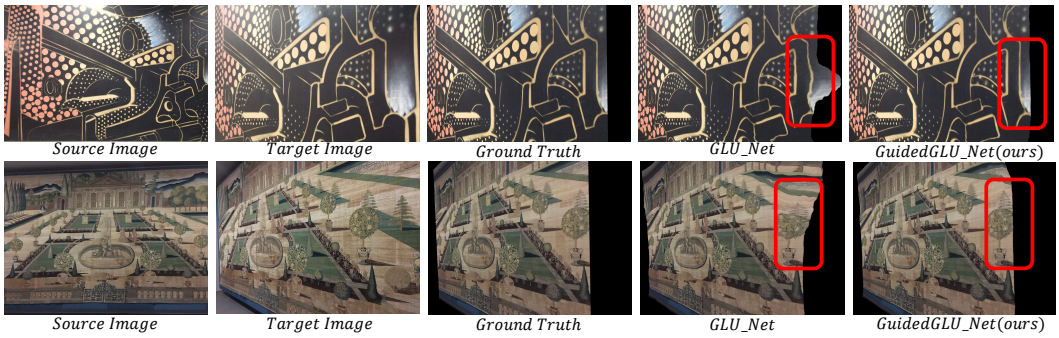


Figure 4.18: Qualitative comparison between GLU-Net and GuidedGLU-Net. Our GuidedGLU-Net is able to find correct correspondences between source and target images with large view-point changes. Note the marked area where GuidedGLU-Net improves the results significantly. (Best viewed in color.)

Chapter 5: Conclusion and Future Work

5.1 Conclusion

In this dissertation, we have studied the deep neural network on 3D point clouds for 3D scene understanding.

First, we proposed a novel approach to perform convolution operation on 3D point clouds, called PointConv. PointConv trains multi-layer perceptrons on local point coordinates to approximate continuous weight and density functions in convolutional filters, which makes it naturally permutation-invariant and translation-invariant. This allows deep convolutional networks to be built directly on 3D point clouds. We proposed an efficient implementation of it which greatly improved its scalability. We demonstrated its strong performance on multiple challenging benchmarks and capability of matching the performance of a grid-based convolutional network in 2D images. Second, to better estimate scene flow directly from 3D point clouds, we proposed a novel learnable cost volume layer along with some auxiliary layers to build a coarse-to-fine deep network, called PointPWC-Net. Because of the fact that real-world ground truth scene flow is hard to acquire, we introduce a loss function that trains the PointPWC-Net without supervision. Experiments on the FlyingThings3D and KITTI datasets demonstrate the effectiveness of our PointPWC-Net and the self-supervised loss function, obtaining state-of-the-art results that outperform prior work by a large margin. As one of the key components in flow estimation networks, the learnable cost volume could be used in various new network designs for flow estimation. Third, we propose a novel point cloud layer, PointConvFormer, which can be widely used in various computer vision tasks. Unlike traditional convolution in which convolutional kernels are functions of the relative position, the convolutional weights of the PointConvFormer are functions of both the relative position and the difference of features. By taking the feature differences into account, the PointConvFormer incorporates the benefits of attention models, which could help the network to focus on points with high feature correlation during

feature encoding. Thorough experiments on a number of point clouds tasks showed that PointConvFormer significantly outperforms traditional point-based operations and outperforms other voxel-based or point-voxel fusion approaches, with significantly less trainable parameters than voxel-based or fusion approaches. Furthermore, we extend the PointConvFormer to the coarse-to-fine matching framework, named GuidedConv. The GuidedConv utilizes coarse predictions as guidance features to filter the convolutional neighborhood for a better finer level prediction. Experiments on dense geometric correspondences from 2D image pairs demonstrate the effectiveness of the GuidedConv over traditional rasterized and continuous convolutions.

5.2 Future Work

5.2.1 Algorithmic Perspective

As a vast research field in computer vision, the dissertation only explores a small fraction of 3D scene understanding from 3D point clouds. In future, there are still plenty of work that could potentially be useful. For PointConv, we proposed an efficient version of pointconv, which largely reduce the memory consumption. However, comparing with traditional convolution, which is highly optimized according the modern hardware, the computation consumption and speed is still slow. There are several ways that might help to improve the computation efficiency. The first one is instead of implement the PointConv with the combination of matmul and 1x1 conv, the pointconv could be directly be implemented in GPU level for better memory management and data transformation. The second optimization could be a better neighbourhood indexing since one of the computation bottleneck for point cloud processing is finding the correct neighbourhood. For PointPWC-Net, the learnable cost volume for 3D point cloud pairs could be used in many different network designs for further improvement. Since the cost volume in PointPWC-Net computes the correlation between patches, the information is more robust and easy to train. It could be used as a novel version of cross-attention for different matching tasks. Finally, the PointConvFormer contains the properties from both convolution and transformer, which unlocks vast potentials in future work. Although the PointConvFormer is designed for 3D point clouds, it could actually be applied to the data in different dimensions, as the PointConv. In future, one of the most important work is

to adopt the pointconvformer to 2D images tasks. To our best knowledge, there has been a operation that combines the properties of convolution and transformer even in 2D deep learning. Besides, the original pointconvformer is proposed on local neighbourhood, it would be interesting to explore the possibilities to work on larger neighbourhoods or even global neighbourhoods.

5.2.2 Application Perspective

One of the goals to study 3D deep learning is to solve real-world problems. With the rapid development of 3D sensors, there are more and more real-world applications. One of the popular applications is autonomous driving, which has drawn more attention these years. Although our algorithms have been evaluated on standard benchmarks in research, we haven't got many chances to explore the application with real-world data. Especially for the scene flow estimation from 3D point clouds, there is almost no real-world data to evaluate on. The benchmark data used now are either from the synthetic 2D data or from the well-calibrated camera systems. In real-world applications, the situation is much more complicated. We would love to experiment with real-world data and further improve our algorithms.

As we know, most deep learning-based methods require vast data to train the model for better performance. However, no dataset in the 3D point cloud could be used as a backbone model training for general purposes. In 2D images, there is ImageNet [26], which contains millions of images for training. In 3D point clouds, there are many datasets for different tasks. Unfortunately, non of the dataset could be served as the ImageNet in 3D point clouds for better training of the 3D models. In future, other than the algorithms on 3D point clouds, it would be useful to explore a large dataset that could be used to train a backbone model for general purposes. As in 2D images, the backbone model could be fine-tuned in different datasets for different computer vision tasks, which could potentially further improve the generalization and robustness of the model.

Lucky, there is an increasing trend in the industry on 3D point clouds processing. With resources in the industry, it would be possible to access much more real-world data and corner cases. It would be interesting to apply 3D algorithms to real data, which could

benefit the whole of humankind. Besides, we believe this thesis is just a small fraction of the exciting and fast-growing field of 3D scene understanding. We hope this thesis could introduce more people to work on the 3D scene understanding topic and would expect to see more future work built upon our work. It would be greatly appreciated to adopt or improve our algorithm for your tasks.

Publication

Wenxuan Wu, Qi Shan, Li Fuxin."PointConvFormer" In Submission.

Zeng, Xianfang, **Wenxuan Wu**, Guangzhong Tian, Li Fuxin, and Yong Liu. "Deep Superpixel Convolutional Network for Image Recognition." IEEE Signal Processing Letters 28 (2021): 922-926.

Li, Xingyi, **Wenxuan Wu**, Xiaoli Z. Fern, and Li Fuxin. "The Devils in the Point Clouds: Studying the Robustness of Point Cloud Convolutions." arXiv preprint:2101.07832 (2021).

Wenxuan Wu, Zhiyuan Wang, Zhuwen Li, Wei Liu, and Li Fuxin. "PointPWC-Net: Cost Volume on Point Clouds for (Self-)Supervised Scene Flow Estimation." **ECCV 2020** Spotlight(Top 5%).

Ziwen Chen, **Wenxuan Wu**, Zhongang Qi, and Li Fuxin. "Visualizing Point Cloud Classifiers by Curvature Smoothing." **BMVC 2020**.

Wenxuan Wu, Zhongang Qi, and Li Fuxin. "PointConv: Deep Convolutional Networks on 3D Point Clouds." **CVPR 2019**, Citation(500+)
Project Web: http://web.engr.oregonstate.edu/ wuwen/pointconv.html

Wenxuan Wu, Li Li, Weiqi Jin. "Disparity refinement based on segment-tree and fast weighted median filter," **ICIP 2016**, Phoenix, AZ, USA, 2016, pp. 3449-3453.doi: 10.1109/ICIP.2016.7533000

Bibliography

[1] Brian Amberg, Sami Romdhani, and Thomas Vetter. Optimal step nonrigid icp algorithms for surface registration. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[2] Lifei Bai, Xianqiang Yang, and Huijun Gao. Nonrigid point set registration by preserving local connectivity. *IEEE transactions on cybernetics*, 48(3):826–835, 2017.

[3] Vassileios Balntas, Karel Lenc, Andrea Vedaldi, and Krystian Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5173–5182, 2017.

[4] Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

[5] Aseem Behl, Despoina Paschalidou, Simon Donné, and Andreas Geiger. Pointflownet: Learning representations for rigid motion estimation from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7962–7971, 2019.

[6] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[7] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9297–9307, 2019.

[8] James R Bergen, Patrick Anandan, Keith J Hanna, and Rajesh Hingorani. Hierarchical model-based motion estimation. In *European conference on computer vision*, pages 237–252. Springer, 1992.

[9] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor*

fusion IV: control paradigms and data structures, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.

[10] Benedict J Brown and Szymon Rusinkiewicz. Global non-rigid alignment of 3-d scans. In *ACM SIGGRAPH 2007 papers*, pages 21–es. 2007.

[11] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *European conference on computer vision*, pages 25–36. Springer, 2004.

[12] Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Lucas/kanade meets horn/schunck: Combining local and global optic flow methods. *International journal of computer vision*, 61(3):211–231, 2005.

[13] Rohan Chabra, Julian Straub, Christopher Sweeney, Richard Newcombe, and Henry Fuchs. Stereodrnet: Dilated residual stereonet. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11786–11795, 2019.

[14] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[15] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

[16] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11030–11039, 2020.

[17] Ran Cheng, Ryan Razani, Ehsan Taghavi, Enxu Li, and Bingbing Liu. 2-s3net: Attentive feature fusion with adaptive feature selection for sparse semantic segmentation network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12547–12556, 2021.

[18] Hung-Yueh Chiang, Yen-Liang Lin, Yueh-Cheng Liu, and Winston H Hsu. A unified point-based framework for 3d segmentation. In *2019 International Conference on 3D Vision (3DV)*, pages 155–163. IEEE, 2019.

[19] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of

indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5556–5565, 2015.

[20] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[21] Xiangxiang Chu, Zhi Tian, Bo Zhang, Xinlong Wang, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Conditional positional encodings for vision transformers. *arXiv preprint arXiv:2102.10882*, 2021.

[22] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[23] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[24] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[25] Martin Danelljan, Giulia Meneghetti, Fahad Shahbaz Khan, and Michael Felsberg. A probabilistic framework for color-based point set registration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1818–1826, 2016.

[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[28] Ayush Dewan, Tim Caselitz, Gian Diego Tipaldi, and Wolfram Burgard. Rigid scene flow for 3d lidar scans. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1765–1770. IEEE, 2016.

[29] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xi-

aohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[30] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[31] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning so (3) equivariant representations with spherical cnns. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–68, 2018.

[32] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[33] Mingliang Fu and Weijia Zhou. Non-rigid point set registration via mixture of asymmetric gaussians with integrated local structures. In *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 999–1004. IEEE, 2016.

[34] Song Ge and Guoliang Fan. Non-rigid articulated point set registration with local structure preservation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 126–133, 2015.

[35] Song Ge, Guoliang Fan, and Meng Ding. Non-rigid point set registration with global-local topology preservation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 245–251, 2014.

[36] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.

[37] Natasha Gelfand, Niloy J Mitra, Leonidas J Guibas, and Helmut Pottmann. Robust global registration. In *Symposium on geometry processing*, volume 2, page 5. Vienna, Austria, 2005.

[38] Nidhi Goyal, Niharika Sachdeva, Anmol Goel, Jushaan Singh Kalra, and Ponnurangam Kumaraguru. Kcnet: Kernel-based canonicalization network for entities in recruitment domain. In *International Conference on Artificial Neural Networks*, pages 157–169. Springer, 2021.

[39] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. *CVPR*, 2018.

[40] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.

[41] Fabian Groh, Patrick Wieschollek, and Hendrik Lensch. Flex-convolution (deep learning beyond grid-worlds). *arXiv preprint arXiv:1803.07289*, 2018.

[42] Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3254–3263, 2019.

[43] Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, and Jianwei Wan. 3d object recognition in cluttered scenes with local surface features: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2270–2287, 2014.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[45] Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. In *SIGGRAPH Asia 2018 Technical Papers*, page 235. ACM, 2018.

[46] Dirk Holz, Alexandru E Ichim, Federico Tombari, Radu B Rusu, and Sven Behnke. Registration with the point cloud library: A modular framework for aligning in 3-d. *IEEE Robotics & Automation Magazine*, 22(4):110–124, 2015.

[47] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[48] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3464–3473, 2019.

[49] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11108–11117, 2020.

[50] Wenbo Hu, Hengshuang Zhao, Li Jiang, Jiaya Jia, and Tien-Tsin Wong. Bidirectional projection network for cross dimension scene understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14373–14382, 2021.

[51] Zeyu Hu, Xuyang Bai, Jiaxiang Shang, Runze Zhang, Jiayu Dong, Xin Wang, Guangyuan Sun, Hongbo Fu, and Chiew-Lan Tai. Vmnet: Voxel-mesh network for geodesic-aware 3d semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15488–15498, 2021.

[52] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 984–993, 2018.

[53] Frédéric Huguet and Frédéric Devernay. A variational method for scene flow estimation from stereo sequences. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–7. IEEE, 2007.

[54] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8981–8989, 2018.

[55] Andrey Ignatov, Nikolay Kobyshev, Radu Timofte, Kenneth Vanhoey, and Luc Van Gool. Dslr-quality photos on mobile devices with deep convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3277–3285, 2017.

[56] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017.

[57] Varun Jain, Hao Zhang, and Oliver van Kaick. Non-rigid spectral correspondence of triangle meshes. *International Journal of Shape Modeling*, 13(01):101–124, 2007.

[58] Varun Jampani, Martin Kiefel, and Peter V Gehler. Learning sparse high dimensional filters: Image filtering, dense crfs and bilateral neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4452–4461, 2016.

[59] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. *Advances in neural information processing systems*, 29:667–675, 2016.

[60] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5010–5019, 2018.

[61] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 66–75, 2017.

[62] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 863–872. IEEE, 2017.

[63] Deyvid Kochanov, Fatemeh Karimi Nejadasl, and Olaf Booij. Kprnet: Improving projection-based lidar semantic segmentation. *arXiv preprint arXiv:2007.12668*, 2020.

[64] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[65] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[66] Abhijit Kundu, Xiaoqi Yin, Alireza Fathi, David Ross, Brian Brewington, Thomas Funkhouser, and Caroline Pantofaru. Virtual multi-view fusion for 3d semantic segmentation. In *European Conference on Computer Vision*, pages 518–535. Springer, 2020.

[67] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019.

[68] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019.

[69] Minhaeng Lee and Charless C Fowlkes. Cemnet: Self-supervised learning for accurate continuous ego-motion estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[70] Huan Lei, Guang Jiang, and Long Quan. Fast descriptors and correspondence propagation for robust global point cloud registration. *IEEE Transactions on Image Processing*, 26(8):3614–3623, 2017.

[71] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3d lidar using fully convolutional network. *arXiv preprint arXiv:1608.07916*, 2016.

[72] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276, 2019.

[73] Ruibo Li, Guosheng Lin, Tong He, Fayao Liu, and Chunhua Shen. Hcrf-flow: Scene flow from point clouds with continuous high-order crfs and position-aware flow embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 364–373, 2021.

[74] Xingyi Li, Fuxin Li, Xiaoli Fern, and Raviv Raich. Filter shaping for convolutional neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[75] Xingyi Li, Wenxuan Wu, Xiaoli Z Fern, and Li Fuxin. The devils in the point clouds: Studying the robustness of point cloud convolutions. *arXiv preprint arXiv:2101.07832*, 2021.

[76] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 2018.

[77] Liang Liu, Guangyao Zhai, Wenlong Ye, and Yong Liu. Unsupervised learning of scene flow estimation fusing with local rigidity. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 876–882. AAAI Press, 2019.

[78] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 529–537, 2019.

[79] Xinhai Liu, Zhizhong Han, Yu-Shen Liu, and Matthias Zwicker. Point2sequence: Learning the shape representation of 3d point clouds with an attention-based sequence to sequence network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8778–8785, 2019.

[80] Min Lu, Jian Zhao, Yulan Guo, and Yanxin Ma. Accelerated coherent point drift

for automatic three-dimensional point cloud registration. *IEEE Geoscience and Remote Sensing Letters*, 13(2):162–166, 2015.

[81] Jiayi Ma, Jun Chen, Delie Ming, and Jinwen Tian. A mixture model for robust point matching under multi-layer motion. *PloS one*, 9(3), 2014.

[82] Jiayi Ma, Junjun Jiang, Huabing Zhou, Ji Zhao, and Xiaojie Guo. Guided locality preserving feature matching for remote sensing image registration. *IEEE transactions on geoscience and remote sensing*, 56(8):4435–4447, 2018.

[83] Jiayi Ma, Ji Zhao, Junjun Jiang, and Huabing Zhou. Non-rigid point set registration with robust transformation estimation under manifold regularization. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[84] Jiayi Ma, Ji Zhao, and Alan L Yuille. Non-rigid point set registration by preserving global and local structures. *IEEE Transactions on image Processing*, 25(1):53–64, 2015.

[85] Jiayi Ma, Huabing Zhou, Ji Zhao, Yuan Gao, Junjun Jiang, and Jinwen Tian. Robust feature matching for remote sensing image registration via locally linear transforming. *IEEE Transactions on Geoscience and Remote Sensing*, 53(12):6469–6481, 2015.

[86] Ningning Ma, Xiangyu Zhang, Jiawei Huang, and Jian Sun. Weightnet: Revisiting the design space of weight networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV 16*, pages 776–792. Springer, 2020.

[87] Ameesh Makadia, Alexander Patterson, and Kostas Daniilidis. Fully automatic registration of 3d point clouds. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 1297–1304. IEEE, 2006.

[88] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. Interpolated convolutional networks for 3d point cloud understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1578–1587, 2019.

[89] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[90] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and

scene flow estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. arXiv:1512.02134.

[91] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4040–4048, 2016.

[92] Iaroslav Melekhov, Aleksei Tiulpin, Torsten Sattler, Marc Pollefeys, Esa Rahtu, and Juho Kannala. Dgc-net: Dense geometric correspondence network. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1034–1042. IEEE, 2019.

[93] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3061–3070, 2015.

[94] Moritz Menze, Christian Heipke, and Andreas Geiger. Joint 3d estimation of vehicles and scene flow. *ISPRS annals of the photogrammetry, remote sensing and spatial information sciences*, 2:427, 2015.

[95] Moritz Menze, Christian Heipke, and Andreas Geiger. Joint 3d estimation of vehicles and scene flow. In *ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015.

[96] Moritz Menze, Christian Heipke, and Andreas Geiger. Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing*, 140:60–76, 2018.

[97] Moritz Menze, Christian Heipke, and Andreas Geiger. Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 2018.

[98] Ajmal S Mian, Mohammed Bennamoun, and Robyn Owens. Three-dimensional model-based object recognition and segmentation in cluttered scenes. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1584–1601, 2006.

[99] Andriy Myronenko and Xubo Song. Point set registration: Coherent point drift. *IEEE transactions on pattern analysis and machine intelligence*, 32(12):2262–2275, 2010.

[100] Thanh Minh Nguyen and QM Jonathan Wu. Multiple kernel point set registration. *IEEE transactions on medical imaging*, 35(6):1381–1394, 2015.

[101] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution net-

work for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.

[102] Chunghyun Park, Yoonwoo Jeong, Minsu Cho, and Jaesik Park. Fast point transformer. *arXiv preprint arXiv:2112.04702*, 2021.

[103] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[104] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing icp variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, 2013.

[105] Gilles Puy, Alexandre Boulch, and Renaud Marlet. Flot: Scene flow on point clouds guided by optimal transport. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII 16*, pages 527–544. Springer, 2020.

[106] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[107] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016.

[108] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[109] Han-Bing Qu, Jia-Qiang Wang, Bin Li, and Ming Yu. Probabilistic model for robust affine and non-rigid point set matching. *IEEE transactions on pattern analysis and machine intelligence*, 39(2):371–384, 2016.

[110] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. *Advances in Neural Information Processing Systems*, 32, 2019.

[111] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial

pyramid network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4161–4170, 2017.

[112] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500*, 2016.

[113] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1164–1172, 2015.

[114] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3577–3586, 2017.

[115] Marcelo Saval-Calvo, Jorge Azorin-Lopez, Andres Fuster-Guillo, Victor Villena-Martinez, and Robert B Fisher. 3d non-rigid registration using color: Color coherent point drift. *Computer Vision and Image Understanding*, 169:119–135, 2018.

[116] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

[117] Jiwon Shin, Rudolph Triebel, and Roland Siegwart. Unsupervised discovery of repetitive objects. In *2010 IEEE International Conference on Robotics and Automation*, pages 5041–5046. IEEE, 2010.

[118] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*, 2017.

[119] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[120] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1746–1754, 2017.

[121] Olga Sorkine. Laplacian mesh processing. In *Eurographics (STARs)*, pages 53–70, 2005.

[122] Hang Su, Varun Jampani, Deqing Sun, Orazio Gallo, Erik Learned-Miller, and Jan Kautz. Pixel-adaptive convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11166–11175, 2019.

[123] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2530–2539, 2018.

[124] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[125] Deqing Sun, Stefan Roth, and Michael J Black. Secrets of optical flow estimation and their principles. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 2432–2439. IEEE, 2010.

[126] Deqing Sun, Stefan Roth, and Michael J Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

[127] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8934–8943, 2018.

[128] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *CVPR*, 2018.

[129] Domen Tabernik, Matej Kristan, and Aleš Leonardis. Spatially-adaptive filter units for compact and efficient deep neural networks. *International Journal of Computer Vision*, 128(8):2049–2067, 2020.

[130] Gary KL Tam, Zhi-Quan Cheng, Yu-Kun Lai, Frank C Langbein, Yonghuai Liu, David Marshall, Ralph R Martin, Xian-Fang Sun, and Paul L Rosin. Registration of 3d point clouds and meshes: A survey from rigid to nonrigid. *IEEE transactions on visualization and computer graphics*, 19(7):1199–1217, 2012.

[131] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *European conference on computer vision*, pages 685–702. Springer, 2020.

[132] Wenbing Tao and Kun Sun. Asymmetrical gauss mixture models for point sets matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1598–1605, 2014.

[133] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent

convolutions for dense prediction in 3d. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3887–3896, 2018.

[134] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European conference on computer vision*, pages 402–419. Springer, 2020.

[135] Pascal Willy Theiler, Jan Dirk Wegner, and Konrad Schindler. Globally consistent registration of terrestrial laser scans via graph optimization. *ISPRS Journal of Photogrammetry and Remote Sensing*, 109:126–138, 2015.

[136] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6411–6420, 2019.

[137] Zhi Tian, Chunhua Shen, and Hao Chen. Conditional convolutions for instance segmentation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 282–298. Springer, 2020.

[138] Prune Truong, Martin Danelljan, and Radu Timofte. Glu-net: Global-local universal network for dense flow and correspondences. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6258–6268, 2020.

[139] Berwin A Turlach. Bandwidth selection in kernel density estimation: A review. In *CORE and Institut de Statistique*. Citeseer, 1993.

[140] Arash K Ushani and Ryan M Eustice. Feature learning for scene flow estimation from lidar. In *Conference on Robot Learning*, pages 283–292, 2018.

[141] Arash K Ushani, Ryan W Wolcott, Jeffrey M Walls, and Ryan M Eustice. A learning approach for real-time temporal scene flow estimation from lidar data. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5666–5673. IEEE, 2017.

[142] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[143] Sundar Vedula, Simon Baker, Peter Rander, Robert Collins, and Takeo Kanade. Three-dimensional scene flow. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 722–729. IEEE, 1999.

[144] Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph

convolutions for 3d shape analysis. In *CVPR 2018-IEEE Conference on Computer Vision & Pattern Recognition*, 2018.

[145] Christoph Vogel, Konrad Schindler, and Stefan Roth. Piecewise rigid scene flow. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1377–1384, 2013.

[146] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, 115(1):1–28, 2015.

[147] Chu Wang, Babak Samari, and Kaleem Siddiqi. Local spectral graph convolution for point set feature learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–66, 2018.

[148] Gang Wang and Yufei Chen. Fuzzy correspondences guided gaussian mixture model for point set registration. *Knowledge-Based Systems*, 136:200–209, 2017.

[149] Jiaqi Wang, Kai Chen, Rui Xu, Ziwei Liu, Chen Change Loy, and Dahua Lin. Carafe: Content-aware reassembly of features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3007–3016, 2019.

[150] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–67, 2018.

[151] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018.

[152] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[153] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. Solov2: Dynamic, faster and stronger. *arXiv e-prints*, pages arXiv–2003, 2020.

[154] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[155] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and

Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):146, 2019.

[156] Yi Wei, Ziyi Wang, Yongming Rao, Jiwen Lu, and Jie Zhou. Pv-raft: Point-voxel correlation fields for scene flow estimation of point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6954–6963, 2021.

[157] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019.

[158] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

[159] Wenxuan Wu, Zhi Yuan Wang, Zhuwen Li, Wei Liu, and Li Fuxin. Pointpwc-net: Cost volume on point clouds for (self-) supervised scene flow estimation. In *European Conference on Computer Vision*, pages 88–107. Springer, 2020.

[160] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[161] Saining Xie, Sainan Liu, Zeyu Chen, and Zhuowen Tu. Attentional shapecontextnet for point cloud recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4606–4615, 2018.

[162] Jia Xu, René Ranftl, and Vladlen Koltun. Accurate optical flow via direct cost volume processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1289–1297, 2017.

[163] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102, 2018.

[164] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5589–5598, 2020.

[165] Zike Yan and Xuezhi Xiang. Scene flow estimation: A survey. *arXiv preprint arXiv:1612.02590*, 2016.

[166] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. *arXiv preprint arXiv:1904.04971*, 2019.

[167] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3323–3332, 2019.

[168] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[169] Li Yi, Hao Su, Xingwen Guo, and Leonidas Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[170] Zhichao Yin and Jianping Shi. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1983–1992, 2018.

[171] Dongdong Yu, Feng Yang, Caiyun Yang, Chengcai Leng, Jian Cao, Yining Wang, and Jie Tian. Fast rotation-free feature-based image registration using improved n-sift and gmm-based parallel optimization. *IEEE Transactions on Biomedical Engineering*, 63(8):1653–1664, 2015.

[172] Julio Zamora Esquivel, Adan Cruz Vargas, Paulo Lopez Meyer, and Omesh Tickoo. Adaptive convolutional kernels. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[173] Feihu Zhang, Jin Fang, Benjamin Wah, and Philip Torr. Deep fusionnet for point cloud semantic segmentation. In *European Conference on Computer Vision*, pages 644–663. Springer, 2020.

[174] Su Zhang, Kun Yang, Yang Yang, Yi Luo, and Ziquan Wei. Non-rigid point set registration using dual-feature finite mixture model and global-local structural preservation. *Pattern Recognition*, 80:183–195, 2018.

[175] Yikang Zhang, Jian Zhang, Qiang Wang, and Zhao Zhong. Dynet: Dy-

namic convolution for accelerating convolutional neural networks. *arXiv preprint arXiv:2004.10694*, 2020.

[176] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10076–10085, 2020.

[177] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16259–16268, 2021.

[178] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019.

[179] Jingkai Zhou, Varun Jampani, Zhixiong Pi, Qiong Liu, and Ming-Hsuan Yang. Decoupled dynamic filter networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6647–6656, 2021.

[180] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In *European Conference on Computer Vision*, pages 766–782. Springer, 2016.

[181] Zhiyong Zhou, Jian Zheng, Yakang Dai, Zhe Zhou, and Shi Chen. Robust non-rigid point set registration using student's-t mixture model. *PloS one*, 9(3), 2014.

[182] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. Cylindrical and asymmetrical 3d convolution networks for lidar segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9939–9948, 2021.

[183] Yuliang Zou, Zelun Luo, and Jia-Bin Huang. Df-net: Unsupervised joint learning of depth and flow using cross-task consistency. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 36–53, 2018.