RWTH AACHEN UNIVERSITY

Diese Arbeit wurde vorgelegt am
Lehr- und Forschungsgebiet Informatik 8 (Computer Vision)
Fakultät für Mathematik, Informatik und Naturwissenschaften
Prof. Dr. Bastian Leibe

Master Thesis

DualConvNet:

Euclidean and Geodesic Convolutions for 3D Semantic Segmentation on Meshes

vorgelegt von

Jonas Schult
Matrikelnummer: 333524
31.01.2020

Erstgutachter: Prof. Dr. Bastian Leibe
Zweitgutachter: Prof. Dr. Leif Kobbelt

Eidesstattliche Versicherung

Jonas Schult	333524
Name	Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Masterarbeit mit dem Titel

DualConvNet: Euclidean and Geodesic Convolutions for 3D Semantic Segmentation on Meshes

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, 31.01.2020	
Ort, Datum	Unterschrift

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten dementsprechend.

Die vorstehende Belehrung habe ich zur Kentnis genommen:

Aachen, 31.01.2020	
Ort, Datum	Unterschrift

Contents

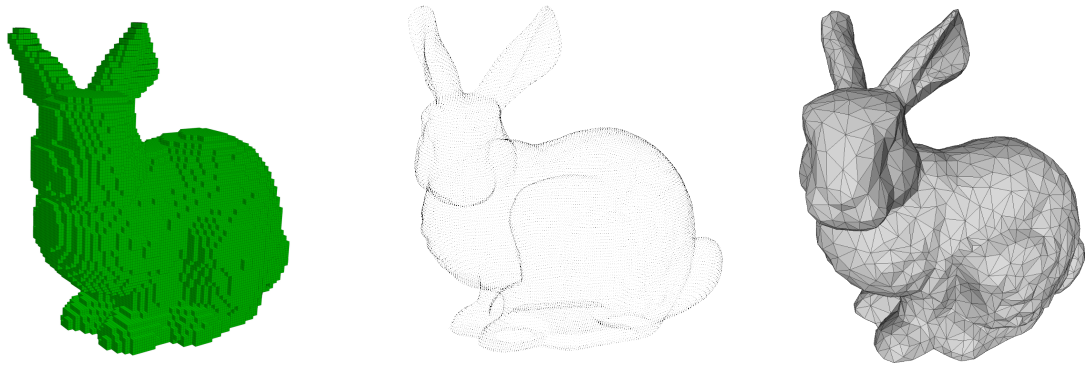
1 Introduction

Recently, we have observed a rise in applications relying on a robust 3D perception of their environment. For instance, domestic assistant robots and autonomous vehicles need to understand their environment in order to operate safely in unknown surroundings. Considering the environment's variety, we resort to data-driven learning-based methods in contrast to explicitly programmed approaches.

Why 3D? The research field of 2D image understanding has shown significant improvements over the past years [HZRS16]. However, a direct adaption of successful 2D approaches does not qualify for 3D perception. The problem lies in the ambiguity of depth estimation using a single monocular camera. Due to the epipolar geometry, corresponding 3D positions of pixels cannot be inferred but are limited to an epipolar line in 3D space for potential correspondences. Thus, 2D images fall short for obtaining an exact 3D representation of the scene. Original methods for directly working on 3D data become necessary. With the availability of 3D sensors such as Kinect, RealSense, and Velodyne, the research field of 3D perception grows significantly lately. Characteristic is their exact depth estimation of objects in the scene.

In computer vision, we have seen the trend that significant improvements have been achieved shortly after releasing challenging large-scale datasets. For example, ImageNet significantly supported the image classification task resulting in well-performing approaches such as ResNets [HZRS16]. Lately, large-scale 3D data sets for autonomous driving tasks have been published containing thousands of training examples of semantically labeled data [CBL+19, KUH+19, SKD+19]. Parallelly to the achievements in 2D perception, we hope that a similar development will take place for 3D environmental understanding, as well.

3D Data Representations. In contrast to ubiquitously used discrete grids for 2D image representation, 3D data representations are diverse and highly depending on the task at hand. In Figure 1.1, we illustrate three commonly encountered representations in the field of 3D learning. A direct extension of 2D images is a voxelized representation



(a) Voxelized [HS17] (b) Point Cloud[1] (c) Mesh [SC14]

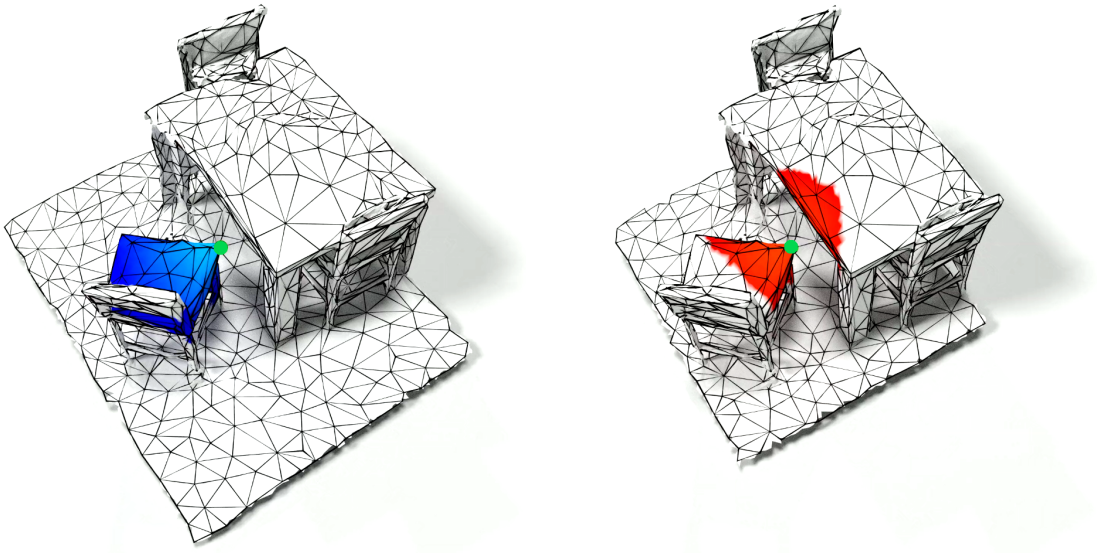
Figure 1.1: **3D Representations.** 3D data is represented in various ways. A voxelized representation makes straight forward adaptions of 2D CNNs possible while introducing information loss due to quantization. Point clouds represent 3D data as a set of points. Although no high-frequency information is lost, new convolutional operators need to be defined in the continuous domain. Contrastingly to the previous ones, meshes additionally contain surface information of 3D objects by edges encoding vertex interconnectivity.

which is illustrated in Figure 1.1a. Here, we place a uniform grid on 3D data points and create a map of occupied grid cells. A real-valued point cloud is therefore reduced to a quantized representation where each occupied cell may be seen as a *volumetric pixel* (thus: voxelized representation). Although introducing a loss of high-frequency information due to the quantization artifacts, this data representation allows a straight forward adaption of 2D discrete convolutions [MS15, GEvdM18, CGS19].

Competingly, numerous works directly operate on continuous point clouds [AML18, HTY18, LBS+18, QSMG17, QYSG17, SJS+18, WQL19, XFX+18]. Here, the data is represented as an arbitrarily sized, unordered set of points (see Figure 1.1b). As we do not voxelize the data, we cannot adapt discrete 2D convolutions. For instance, Wang *et al.* [WYHN18] and Thomas *et al.* [TQD+19] propose convolutional operators directly consuming point clouds. They define convolutional kernels over the Euclidean proximity in terms of k-nn or radius graphs. Since these convolutions solely operate in the Euclidean space and are invariant to any surface structure, we refer to them as *Euclidean* convolutions.

In contrast to the previous representations, meshes represent an enriched data structure. Here, we provide additional surface information encoded by edges interconnecting vertices (see Figure 1.1c). In contrast to point cloud and voxelized approaches, mesh approaches define proximity not in terms of Euclidean k-nn or radius neighbors, but on adjacent mesh neighbors. For instance, Verma *et al.* [VBV18] directly use the mesh-spanning graph to apply graph convolutions on the surface mesh. As these convolutions are independent of any Euclidean proximity and solely operate in graph space, we refer to them as *geodesic* convolutions.

[1] Image source: waldyrious.net/learning-holography/pb-cgh-formulas.xhtml



(a) Geodesic Neighborhood (b) Euclidean Neighborhood

Figure 1.2: **Comparison of Euclidean and geodesic neighborhoods.** Our proposed method DualConvNet combines geodesic and Euclidean convolutions by parallely applying convolutions in the geodesic and Euclidean domain. We assume that geodesic convolutions learn useful feature representations for surface shapes since their receptive field enlarges along the mesh surface. Euclidean convolutions however enable an information flow between geodesically disconnected parts of the scene. Thus, this convolution operator enables to learn feature representation for context information between spatially nearby but geodesically distant objects, such as configurations of objects which usually come in pairs (e.g. chairs with tables). The color gradient shows the geodesic and Euclidean distances between the green center point and its neighbors.

Our contributions. 3D data representations have been studied independently in their main field of application so far. In this thesis, however, we explore the potential of combining *geodesic* and *Euclidean* convolutions on point clouds and meshes simultaneously in the task of 3D semantic scene segmentation. In Figure 1.2, we illustrate our intuition that geodesic and Euclidean convolutions focus on different aspects of feature learning. Geodesic convolutions define proximity in terms of mesh neighbors reachable within k hops on the mesh. When applying convolutions on this neighborhood, we explicitly learn features focusing on the surface structure of the scene. For instance, in Figure 1.2a, the geodesic receptive field of the green center point just comprises vertices of the geodesically close chair surface. It therefore neglects geodesically remote but spatially close vertices of the table. Hence, we assume that geodesic convolutions are more likely to learn feature representations for object shapes.

Contrastingly, Euclidean convolutions focus on the Euclidean proximity of vertices in terms of k-nn or radius neighborhoods in 3D space. They enable an information flow between geodesically disconnected parts of the scene and therefore, we assume

that they learn the interaction between objects. For example, in Figure 1.2b, the convolution does not generate features only based on vertices of the chair but also based on the geodesically remote table. We assume that this contextual information helps to distinguish shape-wise similar classes such as chair and armchair.

Concludingly, we pose the question if a combination of geodesic and Euclidean convolutions brings a significant benefit for the task of 3D scene segmentation. For this purpose, we have developed a simple yet effective multi-scale architecture called *DualConvNet* which combines Euclidean and geodesic convolutions in a parallel manner over multiple scales. Special provisions have been taken to guarantee the modularity of the architecture such that all effects are measurable in order to answer the research question in the ablation study experimentally.

Our approach is *mesh-centric* in that sense that it consumes meshes as a half-edge data structure as well as defining (un-)pooling operations in terms of mesh simplification algorithms. This design decision is necessary to ensure a mesh structure in deeper network layers such that geodesic convolutions are well-defined. We therefore extend *vertex clustering* [RB93] and *Quadric Error Metrics (QEM)* [GH97] as two well-established algorithms from the geometry processing domain such that they can pool and unpool vertex sets from consecutive hierarchy levels. *Pooling Trace Maps* constitute this extension which comprises a look-up dictionary approach for obtaining pooled representatives in constant time. In order to make QEM pooling applicable for large-scale meshes, we finally present a novel sampling strategy for radius neighborhoods called *Random Edge Sampling (RES)* which outputs a sample set which guarantees an upper limit of a predefined expected sample size. In the ablation study, we take a closer look at the properties of RES.

We empirically evaluate our proposed DualConvNet architecture on three publicly available benchmarks for 3D scene segmentation. We achieve competitive results on the ScanNet v2 [DCS+17], as well as the Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [ASZ+16]. Among graph convolutional approaches, we define a new state-of-the-art on both datasets. Moreover, on the recently published Matterport3D benchmark [CDF+17], we can report overall state-of-the-art results. We summarize the main contributions of this thesis as follows: (1) DualConvNet - our novel family of multi-scale convolutional architectures - combines Euclidean and geodesic features for 3D semantic scene segmentation. (2) For creating mesh-centric multi-scale architectures, we extend two well-established mesh simplification algorithms as means of (un-)pooling operations. (3) *Random Edge Sampling (RES)* is a novel sampling method for sampling neighborhoods which guarantees an upper limit for the predefined expected size. Reducing the size of the neighborhood allows us to train networks with substantially fewer neighborhood sizes. However, we infer on test examples with larger sample sizes for better neighborhood approximations. (4) We conclude our work with a thorough ablation study which experimentally proves our claim that Euclidean and geodesic convolutions give a consistent benefit, independent of the pooling method and the notion of the neighborhood used in the architecture.

2 Related Work

In this chapter, we survey related work that purely operates in 3D space. We neglect classical approaches which do not rely on learned features but design feature spaces using hand-crafted features and hybrid methods which also take 2D information such as high-resolution textures into account [HZY+19]. Pure 3D approaches outperform classical and hybrid approaches on a variety of popular benchmarks such that we particularly focus on them in this chapter as they constitute a promising development in this research field.

We motivate the field of 3D (geometric) deep learning focusing on general problems different method categories face. We therefore do not explain architectural design choices in detail since they are independent of global challenges in the field. We highlight the curse of dimensionality (Chapter 2.1.1) and sparsity (Chapter 2.2.1) which particularly become apparent for 3D point cloud data. We show how permutation invariant networks (Chapter 2.1.2) as well as different variations of discrete and continuous convolutions deal with these problem (see Figure 2.1 for a categorized selection of recent publications in this field). We look at the basics of signal processing and the definTition of convolutions in sparse/dense as well as discrete/continuous settings in order to learn locally restricted features (see Chapter 2.2). Moreover, we focus on graph neural networks which leverage the connectivity encoded in meshes in which we are particularly interested in this thesis (see Chapter 2.3).

Current approaches for 2D and 3D understanding rely on multi-scale architectures. We show operations for creating these architectures for traditional grid-based data structures and show how this can be lifted to mesh-based data representations in terms of mesh simplification algorithms (see Chapter 2.4). In a continuous space, there can be an arbitrary number of data points in a certain area. We therefore need to resort to approximate sampling methods to restrict the computational burden an algorithm faces processing these point clouds. Thus, in the last chapter, we present some popular sampling methods for permutation-invariant networks as well as for convolutional approaches (see Chapter 2.5).

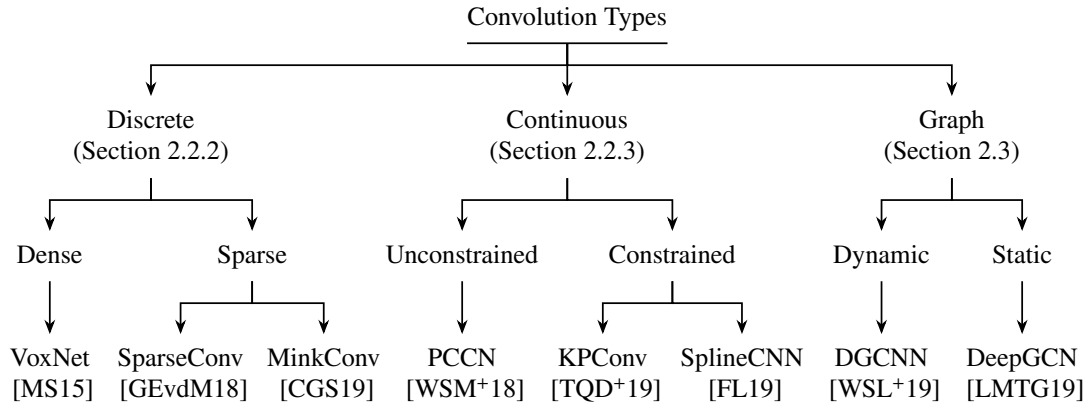


Figure 2.1: **Hierarchy of convolution types**. Recently published methods can be organized in a hierarchy distinguishing between convolutions in the Discrete/Continuous Euclidean space as well as in the graph space. This hierarchy should serve the reader as a road map to the related work content of this thesis pointing to dedicated sections in which the convolution types are presented.

2.1 Learning on Unordered Point Clouds

Approaches operating on 2D images use a discrete 2D grid representation. However, the seminal works on 3D learning by Qi *et al*. [QLJ+17,QYSG17] have arisen the question if another data structure is more suitable for representing 3D point cloud data than quantizing it into a 3D grid representation. They encourage a paradigm shift from discrete grid-like representations to representing data points in an unordered set. Although drastically reducing the amount of memory needed for storing the data (see Chapter 2.1.1), we need to define new permutation-invariant operators (see Chapter 2.1.2) to learn useful features on sets of points which can deal with their unordered nature.

2.1.1 The Curse of Dimensionality

Traditional approaches for operating on 3D data quantize a sparse point cloud to form a dense 3D grid where each cell comprises either a feature vector or a null vector for representing unoccupied cells [MS15]. Their inherent challenge is the vast amount of blank cells which do not contribute to prediction tasks. Applying usual 3D convolutions result in a significant computational overhead in regions where little valid information is present. Previous deep learning methods have been mostly applied on 2D representations, the effect of unoccupied regions becomes even more critical for 3D data due to the *curse of dimensionality*.

The curse of dimensionality is a general term in computer science and mathematics used to describe similar problems in e.g. machine learning, dynamic programming or combinatorics [KM17]. The characteristic feature among these areas is the problem that the complexity of the algorithm increases exponentially with the dimensionality of the input data.

We illustrate this in the following example adapted from Keogh *et al*. [KM17]: in a unit length quadratic surface embedded in 3D space, we uniformly place 9 data points. The surface is now evenly sampled with a minimal distance of 1/3 between data points. When lifting this surface to a unit-length cubical volume in 3D space, we now can uniformly place 27 data points in this volume while still guaranteeing the minimal distance requirement of 1/3 between data points. However, when dealing with the same number of data points (here: 9) the minimal distance between them will increase with the number of dimensions which means that more space will be unoccupied. In the application field of 3D learning on sparse point clouds, this means that most computations for discrete 3D convolutions are made on unoccupied areas in space which increases the inference time of the algorithm as well as introducing a huge memory consumptions. Therefore, traditional approaches such as VoxNet [MS15] are not able to build deep and complex architectures for point cloud learning using dense voxelized representations.

2.1.2 Learning Permutation-Invariant Point Representations

The seminal work of PointNet introduced by Qi *et al*. [QSMG17] first explored the possibility to directly learn features on point cloud representations rather than generating hand-crafted features or discretizing the data into dense grid representations. Considering the curse of dimensionality described in the previous Chapter, PointNet aims at overcoming this problem by a paradigm shift towards point-based approaches. However, this approach faces its own challenges. It is not possible to feed an unordered set of points to a network. Here, the ordering matters and special arrangements have to be taken to ensure the *permutation-invariance* of the neural network. Different strategies exist to guarantee this property: (1) The input data is transformed into a canonical ordering. However, in the general case, it is mathematically not possible to find a bijective mapping from a high-dimensional feature space to an ordered one-dimensional representation. (2) Another strategy is to leverage recurrent neural networks and train them with permutated sequences. We need to augment the input data consisting of N points to generate all $N!$ permutations on them. Thus, the training time will drastically increase. However, Vinyals *et al*. [VBK16] show that the order indeed matters even for recurrent neural networks. (3) The final approach which was used by Qi *et al*. [QSMG17], leverages *symmetric* aggregation functions. Symmetric aggregation functions take arbitrarily many features into account and always output the same result independent of the ordering of features. Simple symmetric functions include $\max(\cdot)$, $\min(\cdot)$, $\text{mean}(\cdot)$, $+$ and $*$. However, applying a symmetric function directly on the point cloud does not extract useful features. Therefore, PointNet learns a permutation-invariant function f which symmetrically aggregates the learned features of the point set lifted into a higher-dimensional space by an MLP function h. They prove that the function $f(\{x_1, \dots, x_n\}) \approx g(h(x_1), \dots, h(x_n))$ is theoretically able to approximate any symmetric function. Here, $f : \mathcal{P}(\mathbb{R}) \to \mathbb{R}$ is a learned symmetric function consisting of a MLP $h : \mathbb{R} \to \mathbb{R}^K$ which maps a feature into a K dimensional feature space. The simple symmetric aggregation function $g : \mathcal{P}(\mathbb{R}^K) \to \mathbb{R}$ maps the high-dimensional feature representation of the point back to a scalar representation.

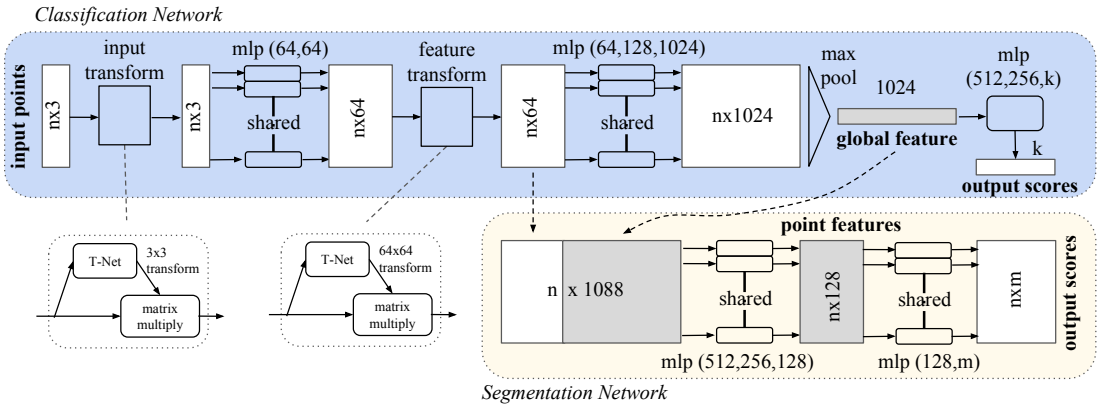


Figure 2.2: **PointNet architecture for classification and semantic segmentation.** Each point of a set containing n points is independently transformed to a 1024 dimensional feature representation. Using the symmetric $\max(\cdot)$, we obtain a global feature representation of the point set. For classification tasks, this representation is MLP-transformed to predict the final class whereas it is concatenated with local point features for semantic segmentation tasks. This figure is adopted from [QLJ+17].

In Figure 2.2, we show PointNet's architecture for classification and semantic segmentation. The sequence of shared MLPs transforms each point independently to a 1024 feature representation. Note that this behavior is analog to the h function described above. Applying the symmetric $g = \max(\cdot)$ function on all n features leads to a permutation-invariant global feature representation of the full point cloud.

The particular limitation of the PointNet architecture is their bottleneck of understanding the global context by max-pooling point features. This is the only step in the network where features of points within the scene are fused which poses a limitations for learning features considering the locality of points. Moreover, in order to describe a full scene just within one single feature vector, this vector needs a reasonably large size. For the original PointNet, the global representation contains 1024 features to which every point representation has to be lifted before aggregation. Thus, PointNets are not applicable for directly operating on large-scale point clouds for semantic scene segmentation. Qi *et al.* [QLJ+17, QYSG17] overcome this problem by cropping $1\text{m} \times 1\text{m}$ blocks out of the point cloud and segment them individually. Thus, PointNets are applied in a sliding-window fashion which has a negative influence on both the runtime and capturing the global context.

2.2 Learning Local Contexts in Euclidean Space

Unlike permutation-invariant networks, numerous alternative approaches rely on convolutions [GEvdM18, TQD+19, FL19, CGS19, HRV+18, WSM+18, EKL20]. The input data is interpreted as signals on which a filter kernel is convolved. Convolutions have proven to work well as local feature extractors in tasks such as 2D/3D understanding while using significantly fewer parameters than the size of the input (*parameter sharing*) and being robust to translations in the input domain (*translation equivariance*).

Formal definition. First, we present the general *continuous* convolution form and motivate its specific properties:

$$\big(f * g\big)(x_i) = \int_{-\infty}^{+\infty} f(x_j) \odot g(x_i - x_j)\, dx_j \tag{2.1}$$

The continuous feature function $f : \mathbb{R}^3 \to \mathbb{R}^F$ assigns a feature vector to every 3-dimensional position $x_j \in \mathbb{R}^3$ and the continuous kernel function $g : \mathbb{R}^3 \to \mathbb{R}^F$ maps a relative position to its kernel weight. Calculating the Hadamard product $\odot$ (elementwise multiplication) of the feature vector and the mirrored kernel function centered around x_i results in a moving average weighted by the filter function g. The *sliding window* behavior of the kernel leads to interesting technical properties of convolutions. Among them, *translation equivariance* as well as *parameter sharing* are explained in the next paragraphs.

Translation equivariance. The property of *translation equivariance* states that applying a convolution on a signal translated by $\boldsymbol{T}(\cdot)$ equals to translating the convolved signal by $\boldsymbol{T}(\cdot)$ [GBC16]. This implies that the convolution with the kernel function g of an arbitrary feature pattern at a specific location always returns the same output neglecting the absolute features' position.

$$\big(\boldsymbol{T}(f) * g\big)(x) = \big(f * g\big)(\boldsymbol{T}(x)) \tag{2.2}$$

Note that convolutions are not *translation-invariant* because after a translation the convolved features will be positioned in another location. This leads to the observation that the extracted features are independent of their absolute position in space. For instance, this property becomes very handy in the following example. The features for an object should not depend on their absolute position in the scene but rather be defined over locally extracted features and their relative position to other objects' features.

Parameter sharing. A direct consequence of the convolution's translation equivariance is *parameter sharing*. The weights of the kernel are reused at different positions of the input signal, thus drastically reducing the size of the parameter space and decoupling it from the size of the input signal. This phenomenon is called *parameter sharing* and constitutes a crucial improvement over fully-connected approaches, since it is generally possible to model a filter kernel with a comparably lower number of parameters than the size of the input data.

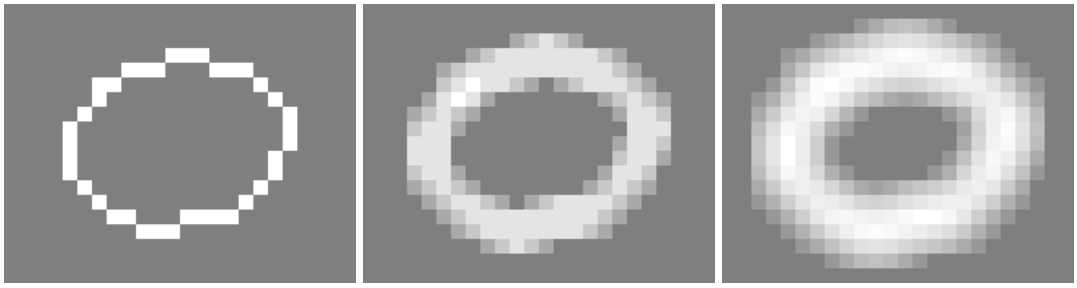


Figure 2.3: **Submanifold Dilation Problem.** The illustrated submanifold (here: circle) gets gradually dilated by consecutively applying a dense 3×3 convolution. With each convolution, more features blur into undefined areas and break the *feature sparsity*. The figure is adopted from [GEvdM18].

2.2.1 The Curse of Sparsity

In Chapter 2.2, we have motivated convolutions as local feature extractors for continuous signals. However, in practical applications, we do not have associated feature vectors for all positions in the 3D space. Thus, we deal with a *sparse* representation of the scene. Here, the general form of continuous convolutions in Equation 2.1 cannot be applied anymore since the Hadamard product is undefined on sparse feature vectors.

Thus, multiple works [HRV+18, WSM+18, EKL20] leverage approximate methods. They interpret existing data points as samples drawn from an underlying distribution. Then, Monte Carlo integration is used to approximate the convolution with K drawn samples:

$$\big(f * g\big)(x_i) \approx \frac{1}{K} \sum_{k=1}^{K} f(x_k) \cdot g(x_i - x_k) \tag{2.3}$$

We refer to Chapter 2.5.2 for more information about sampling the Euclidean neighborhood space.

Submanifold Dilation Problem. An interesting challenge evolves out of the aforementioned sparsity and dimensionality problem. In our task of semantically segmenting scenes, our input data comprises point clouds sampled from objects' surfaces obtained by 3D sensors. These sampled surfaces constitute 2D submanifolds embedded in 3D space. Naively applying convolutions over all the sparse space leads to a blurring out effect of *active points* (here: points with a valid associated feature vector) to undefined regions in space. In the general case, this dilation does not necessarily pose a problem and is rather desirable. 2D images operate in a *dense* 2D discrete grid space. Here, we want that features of pixels blur into neighboring regions, thus increasing the receptive field and enabling a prediction not only based on local decisions.

However, in sparse settings, this behavior is undesirable. With each application of convolutions, active points blur into undefined regions and thus increasing the number of data points which have to be considered in the next hidden layer. Thus, the size of active points rapidly increases throughout the network and becomes computationally infeasible. This phenomenon is called *submanifold dilation problem* [GEvdM18] and is

illustrated in Figure 2.3. For volumetric approaches, the resolution of the quantization grid bounds this expansion of data points. However, for continuous convolutions, it is unbounded and thus not applicable at all.

A simple solution to the submanifold dilation problem is to restrict the space convolutions are applied to. Here, convolutions are only allowed to be applied at points where valid feature vectors are present [GEvdM18, WSM+18, EKL20, TQD+19]. Special provisions have to be taken in order to ensure that the receptive field is not only limited to connected components but can reach throughout all the space. Chapter 2.2.2 deals with specific solutions for this challenge.

Enforcing Kernel Locality. Restricting the kernel to be only locally defined around a center point leads to a limited number of points considered for the convolution. This results in some interesting technical properties described in the next paragraph.

First, the computational load is reduced. Instead of considering an unbounded number of points for the convolution, locally restricted convolution kernels only consider surrounding regions. Moreover, this explicitly enforces *local feature learning*. Stacking these convolutions leads to an increased receptive field. Volumetric approaches control the locality with the size of their discrete kernels analogous to 2D convolutions [GEvdM18]. Contrastingly, continuous convolutions resort to various approaches of neighborhood searches [HRV+18, TQD+19, WSM+18]. We refer to Chapter 2.2.3 for more details on neighborhood notions for continuous convolutions.

2.2.2 Discrete Kernels for Sparse Signals

In the previous chapter, we have motivated learnable convolutions in their general form. In this chapter, we focus on discrete convolutions as one instantiation of them. In order to make them applicable, we firstly have to quantize the continuous space into regular cubic volumetric pixels (*voxels*). This step can be seen as a special kind of pooling which aggregates point cloud information and enables a more compact data representation by low-pass filtering the input data. This data representation is similar to *dense* 2D image representations which are de-facto standard for 2D image understanding. As explained in Chapter 2.1.1, the curse of dimensionality makes it, however, computationally infeasible to deal with sparse large-scale scenes in a dense grid-like fashion. Methods like [MS15] suffer from high-memory consumption and thus, are not able to build complex enough models to achieve state-of-the-art results.

In this chapter, we focus on sparse discrete convolutions that extend standard convolutions from dense to sparse grids and achieve remarkable results with a low memory footprint. *Active sites* constitute the technical contribution that enables convolutions to be applied in that manner. An active site marks a voxel to contain a valid feature vector. In order to efficiently apply convolutions on the input data, sparse hash maps are used to map active sites (voxel positions) to their corresponding feature vectors. As a representative of this category, we present *submanifold sparse convolutions* [GEvdM18] which currently achieve competitive results in various 3D semantic segmentation benchmarks.

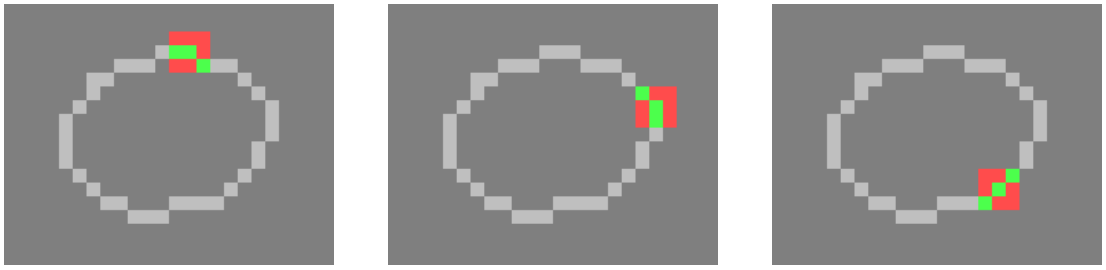


Figure 2.4: **Submanifold Sparse Convolution (SSC).** SSCs are only applied centered around active sites (green). For the convolution, just active sites are used and inactive sites (red) are ignored, e.g. set to 0. Thus, the number of active sites stay constant throughout the network and prevent the submanifold dilation problem (see Figure 2.3). The figure is adopted from [GEvdM18].

(Submanifold) Sparse Convolutions. The idea of Graham *et al.* [GEvdM18] consists of two parts: ① Due to the curse of dimensionality, convolutions are only applied if at least one active site is in the proximity of the kernel (*sparse convolution*), ② With respect to the submanifold dilation problem, convolutions are only applied on positions centered around active sites (*submanifold sparse convolution*).

In order to tackle requirement ①, sparse convolutions are introduced. Sparse convolutions are only applied at those positions in space where the discrete filter kernel covers at least one active site. Here, we significantly reduce the computational load. Instead of convolving over the entire space of the scene which mostly consists of inactive sites, we only consider regions where data is present. If an inactive site is multiplied with a kernel weight, it is set to 0 on-the-fly. The idea here is analogous to Equation 2.3 introduced in Chapter 2.2 and can be seen as a sampling method of the input data.

However, this approach does not solve the submanifold dilation problem. If the sparse convolution is centered around an inactive site and active sites are in the proximity of the filter kernel, we introduce a new active site for the next hidden layer (see Figure 2.3). Since we stack many convolutions in modern neural network architectures, we significantly increase the number of active sites throughout the network. This eventually results in a vast computational load which makes sparse convolutions not applicable for deep neural network architectures (compare submanifold dilation problem in Chapter 2.2.2). Prerequisite ② deals with this aforementioned problem. Submanifold sparse convolutions differ from sparse convolution in that they are only applied to already active sites (see Figure 2.4). Applying submanifold sparse convolutions introduce challenges regarding the size of the receptive field. In comparison with sparse convolutions, submanifold convolutions are only capable of propagating information within a single connected component. Since the submanifold is not dilated (blurred out), we cannot exchange any information between disconnected components. As a practical example, it might be helpful for the network to know that the nearby object is a desk in order to distinguish between an ordinary chair and a desk chair. Thus, submanifold sparse convolutional networks heavily rely on well-set pooling and strided convolutional layers in order to bridge between connected components and propagate information among them.

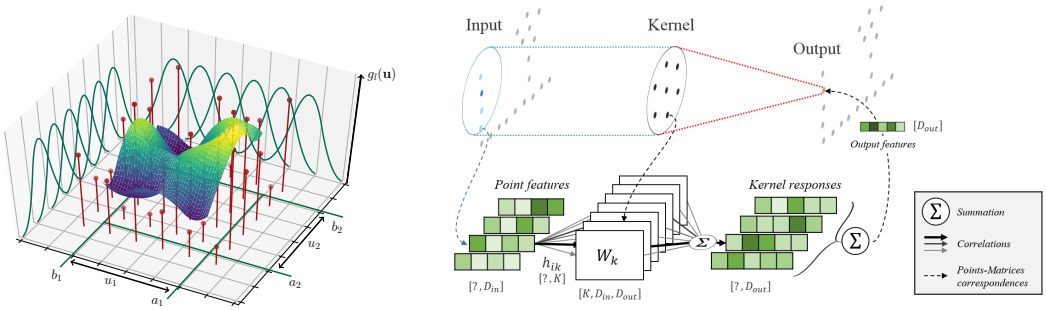
Active	Type	C	SC	SSC
Yes	FLOPs	$3^d mn$	amn	amn
	Memory	n	n	n
No, $a > 0$	FLOPs	$3^d mn$	amn	0
	Memory	n	n	0
No, $a = 0$	FLOPs	$3^d mn$	0	0
	Memory	n	0	0

Table 2.1: **Computational efficiency of convolutional operators.** We compare dense convolutions (C), sparse convolutions (SC), and submanifold sparse convolutions (SSC) in terms of calculation steps and memory consumption. The convolution is applied in d-dimensional space with a kernel size of $f = 3$, padding $p = 1$, m input features and n output features. The number of actives sites in the proximity of the convolution is denoted as a. We adopt this comparison from Graham *et al*. [GEvdM18].

Computational efficiency. In Table 2.1, we compare the computational efficiency of dense convolutions, sparse convolutions and submanifold sparse convolutions in terms of number of calculation steps and memory consumption. Using a discrete filter kernel with side length $f = 3$, we end up needing $3^d mn$ calculation steps for d dimensions, m input features and n output features for dense convolutions since they do not differentiate between active and inactive sites. The memory consumption always equals the size of the output feature vector. Since (submanifold) sparse convolutions operate on the active site space, the number of calculation steps equals amn with the number of active sites in the proximity of the calculation $a \leq 3^d$. We already see a clear performance improvement of (submanifold) sparse convolutions over dense convolutions. Moreover, in special cases such as an inactive center site or no active sites in the convolution's proximity, the calculation steps and memory consumption fall to 0 which constitutes a significant advantage of sparse convolutions or dense convolutions.

2.2.3 Continuous Kernels for Sparse Signals

In the previous Chapter 2.2.2, we have defined discrete kernels for sparse signals. In order to be applicable, discrete kernels need a grid-structured representation. This quantization constitutes a low-pass filtering of the input data. We, thus, deal with a loss of information which might lead to worse segmentation results. In this chapter, we present an orthogonal approach to discrete convolutions. Continuous convolutions for sparse signals can be applied to raw point clouds without the inevitable loss of high-frequency information. Here, the challenge consists in defining efficient continuous kernels in terms of computation and memory consumption as well as introducing countermeasures for the curse of dimensionality and the submanifold dilation problem.



(a) Quadratic B-Splines used by SplineConv [FL19]

(b) kernel points for aggregating features of neighboring points as presented in KPConv [TQD+19]

Figure 2.5: SplineConv [FL19] and KPConv [TQD+19] represent two variations of constrained implicit kernel representations where the learnable parameters θ are used to parameterize kernel functions with desirable properties.

Implicit kernel functions. In Equation 2.3, we have defined an interpretation of Monte Carlo integration for continuous convolutions for K samples drawn from the neighborhood distribution. We have left it open how the kernel function g is defined. That is how the kernel function maps relative positions to their corresponding kernel weights. In the case of discrete convolutions, g is defined as a simple look-up operation which maps neighboring voxel positions to their kernel weights stored in a matrix. However, in the continuous case, we deal with infinitely many neighboring positions and thus cannot define *explicit* look-up operations due to the infinite memory consumption. In order ease the memory consumption problem, we resort to *implicit* kernel functions:

$$g(p; \theta) = \phi(p; \theta) \tag{2.4}$$

where the kernel function g maps a relative position $p = x_i - x_k \in \mathbb{R}^d$ to its corresponding kernel weights using a function $\phi : \mathbb{R}^d \to \mathbb{R}^d$ parameterized by learnable parameters θ. In the same manner as the submanifold dilation problem is solved in submanifold sparse convolutions, continuous convolutions solve the problem, as well. Instead of applying the convolutional kernel at all location in the space, it is only applied at points with corresponding feature vector which results in a stable number of points for subsequent layers.

(Un)constrained kernel functions. The learnable function ϕ from Equation 2.4 has been subject to various works on continuous convolutions recently. As illustrated in Figure 2.1, methods can be separated in two distinct groups: unconstrained [EKL20, WSM+18] and constrained [TQD+19, FL19, MBM+17, XFX+18] implicit kernel functions. In the unconstrained case, ϕ is defined as a multi-layer perceptron *directly* operating on relative point positions.

In the constrained case, ϕ is defined as a function where some of the parameters are learnable. Here, the design of these functions can be influenced in order to ensure specific properties of this convolution, e.g. kernel weights following a Gaussian Mix-

ture Model [MBM+17] or B-spline function [FL19] (see Figure 2.5a). Generally, it is possible to build up arbitrarily complex kernel functions with significantly less learnable parameters which ease the training process of such convolutions. In Figure 2.5, we show the constrained kernel definitions of SplineConv [FL19] and KPConv [TQD+19].

2.3 Learning on Graphs

In Chapter 2.1.2 and Chapter 2.2, we have presented permutation-invariant networks and convolutions on sparse signals. However, in this chapter we focus on an orthogonal approach representing 3D data as graph structures. Similar to the work of Fey *et al*. [FL19], a graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{I})$, whereas the vertex-feature matrix $\mathcal{V} \in \mathbb{R}^{N \times f}$ contains row-wise vertices with f-dimensional feature vectors each. These vertices are interconnected via an edge-index matrix $\mathcal{I} \in \mathbb{N}_{\leq N}^{E \times 2}$ of E edges where each row encodes the directed edge relationship between two vertices. Moreover, the edge-feature matrix $\mathcal{E} \in \mathbb{R}^{E \times d}$ encodes d-dimensional edge features in a row-wise fashion.

Graphs are ubiquitously used in a variety of different research fields not limited to the field of computer vision. For further reading, we like to refer to the work of Kipf *et al*. [KW17] and Gilmer *et al*. [GSR+17] who use graph representations to approach node classification problems on citation graphs or predicting molecular properties in protein-protein interaction networks.

Meshes represent discrete manifolds embedded in 3D space. In our work, we particularly deal with meshes which are instantiations of graphs embedded in 3D space. Using 3D sensors, we obtain a point cloud of N points sampled from objects' continuous surfaces, e.g. 2D manifolds embedded in 3D space. This continuous surfaces are transformed to *discrete* manifolds by meshing the N sampled points to generate interconnecting faces $\mathcal{F} \in \mathcal{V}^3$ between vertices [BBL+17]. The edge-index matrix $\mathcal{I}$ is simply induced by the set of faces $\mathcal{F}$ resulting in $2 \cdot 3$ directed edges per face.

We particularly benefit from the mesh's ability to encode surface information by edges between vertices. We can therefore learn surface-dependent features on the mesh structure to represent the inherent three-dimensional shape of objects.

2.3.1 Graph Convolutional Neural Networks

The substantial advantage of graph convolutions over Euclidean convolutions (Chapter 2.2) is the less restricted way of modeling relationships between data points. Here, the edge-index matrix $\mathcal{I}$ is capable of interconnecting arbitrary vertices whereas Euclidean convolutions are bounded to local neighborhoods in the Euclidean domain. Most importantly, Euclidean convolutions even operate on vertices sampled from other manifolds, e.g. another object in the scene. Thus, they wrongly fuse information which lead to corrupted surface-dependent features.

Definition. In the following paragraph, we discuss the general form of graph convolutional neural networks which is adapted from Fey *et al.* [FL19]:

$$v_i' = \overbrace{\gamma(v_i, \underbrace{\square_{j \in \mathcal{N}(i)}}_{\text{aggregation } \mathcal{A}_i} \overbrace{\phi\left(v_i, v_j, e_{j,i}; \theta\right)}^{\text{message } m_{j,i}}; \psi)}^{\text{update } v_i \text{ with } \mathcal{A}_i} \tag{2.5}$$

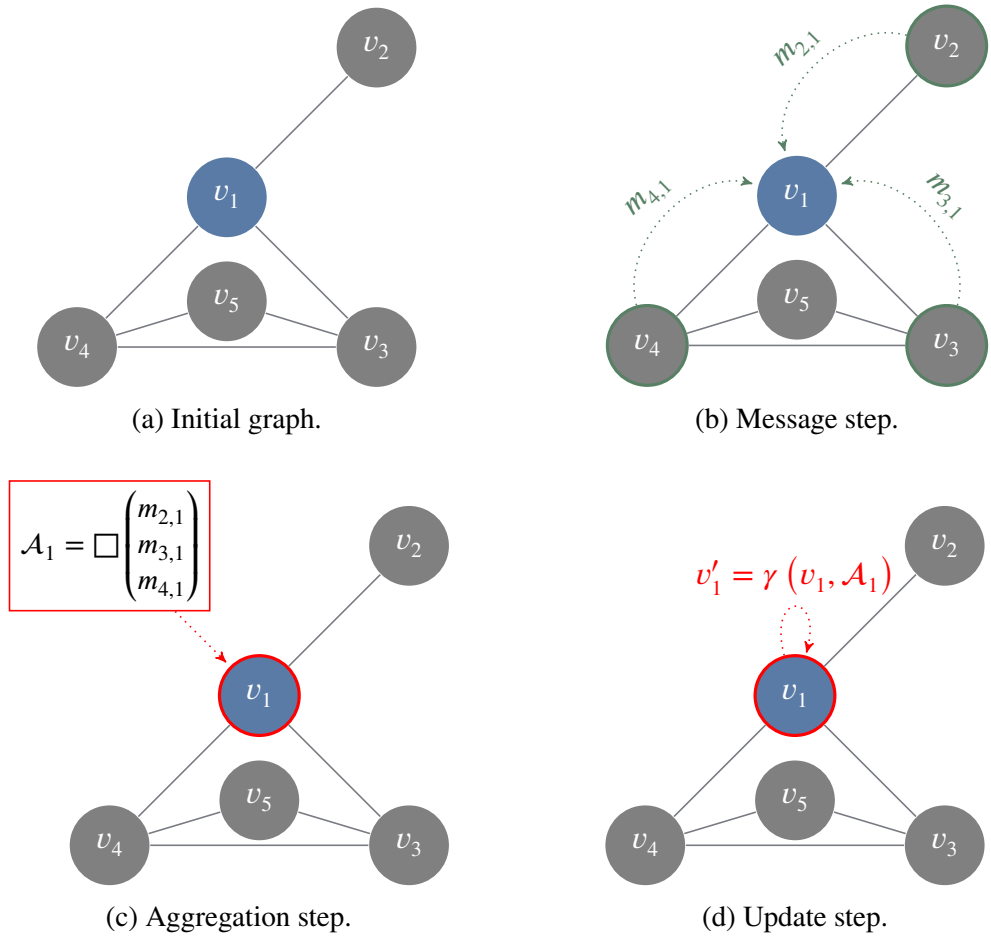
A graph convolutional layer basically consists of three steps: *message*, *aggregation* and *update* step. Because of this behaviour it is often referred to as a *message passing algorithm* where messages are generated, send over the directed edges of the graph, aggregated at their receiving vertices and fused with the current state of the vertex. It is considered *convolutional* since the learnable parameters θ and ψ are shared among all vertices and edges, respectively. In the next paragraphs, we shed light upon all three components of graph convolutions.

Message step. In graph convolutional neural networks, the differentiable function $\phi\left(v_i, v_j, e_{i,j}; \theta\right) \in \mathbb{R}^h$ generates messages (h-dimensional feature vectors) for each edge $(i, j) \in \mathcal{I}$ of the graph $\mathcal{G}$. It takes the vertex feature representations of v_i and v_j as well as the edge feature $e_{i,j}$ into account and learns a message feature with trainable parameters θ. Intuitively, these messages are sent along the directed edges to the their corresponding receiving vertices v_i (see Figure 2.6b).

Here, a difference is established between Euclidean convolutions and graph convolutions. In Equation 2.3, the convolutional kernel receives the relative position between the center point and a neighboring point *in the Euclidean domain*. Contrastingly, in Equation 2.5, graph convolutions operate *in the graph space* where neighboring vertices can be arbitrarily distant. Thus, the receptive field increases in the graph domain and not in the Euclidean domain. Consequently, by propagating messages, with each applied graph convolution, the graph is explored in a 1-hop fashion. Moreover, the Euclidean convolution kernel g generates kernel weights according to *relative Euclidean* position, whereas the graph convolutional kernel ϕ takes the *feature representation* of vertices v_i and v_j into account. Locality in graph convolutional neural networks is thus defined in terms of the graph feature space rather than the Euclidean domain.

Aggregation step. From incoming edges, each vertex receives a arbitrary number of messages $m_i = \left(m_{i,j}, \ldots, m_{i,M_i}\right)^T \in \mathbb{R}^{M_i \times h}$. The size M_i differs for each vertex i and is dependent on its number of neighbors. Therefore, aggregating incoming messages to a fixed-size representation poses a challenge and makes fully-connected or convolutional approaches inapplicable.

We resort to permutation-invariant aggregation functions $\mathcal{A} = \square(\cdot)$, since the varying number and the ordering of messages should not influence the aggregated feature vector (see Figure 2.6c). Non-learned functions include $\max(\cdot)$, $\min(\cdot)$ and $\text{mean}(\cdot)$ or even PointNets. The dedicated Chapter 2.1.2 deals with permutation-invariant functions in more detail.



(a) Initial graph. (b) Message step.

(c) Aggregation step. (d) Update step.

Figure 2.6: **Visualization of a single graph convolution.** A graph convolution consists of a message (b), aggregation (c) and update (d) step. Messages are past along the edges (b), at their receiving vertices aggregated in a permutation-invariant fashion (c) and subsequently the state of the current vertex is updated by fusing the old and new features (d). For illustration purposes, we show all steps just for vertex v_1.

Update step. The final step of a graph convolution consists of updating the feature representation v_i of vertex i. The differentiable function $\gamma\left(v_i, \mathcal{A}_i; \psi\right)$ with learnable parameters ψ learns to fuse old vertex features with the aggregated features of the receiving messages. Figure 2.6d illustrates the step.

Efficiency. In Algorithm 1, we present a parallized version for graph convolutional neural networks. Each major step (message, aggregation and update) can be performed in parallel and thus utilizes the GPU efficiently. Specialized libraries such as PyTorch Geometric [FL19] or Graph Nets [BHB+18] provide implementation for leveraging graph neural networks for GPUs.

Algorithm 1: Message Passing in Graph Neural Networks

Input : Initial graph $\mathcal{G} = \left(\mathcal{V}^{(0)}, \mathcal{E}, \mathcal{I}\right)$ with vertex feature matrix $\mathcal{V}^{(0)} \in \mathbb{R}^{N \times f^{(0)}}$ of N vertices with f features at step $k = 0$, edge index matrix $\mathcal{I} \in \mathbb{N}_{\leq N}^{E \times 2}$ of E edges and edge features $\mathcal{E} \in \mathbb{R}^{E \times d}$ with d dimensional edge features

Output: vertex feature matrix $\mathcal{V}^{(k)} \in \mathbb{R}^{N \times f^{(k)}}$ after k steps on the graph $\mathcal{G}$

while $k < K$ **do** `// perform K message passing steps`
  $k \leftarrow k + 1$;
  foreach *edge* (i, j) *in parallel* **do**
    $m_{i,j} \leftarrow \phi^{(k)}\left(\mathcal{V}_i^{(k-1)}, \mathcal{V}_j^{(k-1)}, \mathcal{E}_{i,j}; \theta\right)$; `// message step`
  end
  foreach *vertex* i *in parallel* **do**
    $a_i \leftarrow \square_{j \in \mathcal{N}(i)} m_{i,j}$; `// aggregation step`
    $\mathcal{V}_i^{(k)} \leftarrow \gamma^{(k)}\left(\mathcal{V}_i^{(k-1)}, a_i; \psi\right)$; `// update step`
  end
end

2.3.2 Dynamic Graph CNN

As a representative of graph convolutional neural networks, we present Dynamic Graph CNNs introduced by Wang *et al.* [WSL+19]. Its unique appeal lies in the dynamical recalculation of graphs' k-nn neighborhoods in various latent feature spaces. In contrast to Euclidean convolutions, this breaks the Euclidean locality of feature learning and allows to group and refine semantically similar points in the feature space.

EdgeConvs interpreted as GCN layers. In this paragraph, we show how EdgeConvs are classified into the graph convolutional framework. In the following, we thus define the necessary message, aggregation and update function. The message function ϕ for EdgeConvs are defined as follows:

$$\phi\left(v_i, v_j, e_{j,i}; \theta\right) = \varphi\left([v_i, v_j - v_i]; \theta\right) \tag{2.6}$$

φ denotes an multi-layer perception parameterized by θ with an arbitrary number of hidden feature representations (interlinked with ReLU activation functions). The input comprises concatenated features of the current vertex i as well as the difference of the features of neighboring vertices j and the current vertex i. Therefore, the directed edge is explicitly modelled by the difference of feature vectors which gives the operator its name *EdgeConv*. For aggregating messages, DGCNN relies on a simple $\square = \max(\cdot)$ permutation-invariant function in order to obtain the most pronounced feature per di-

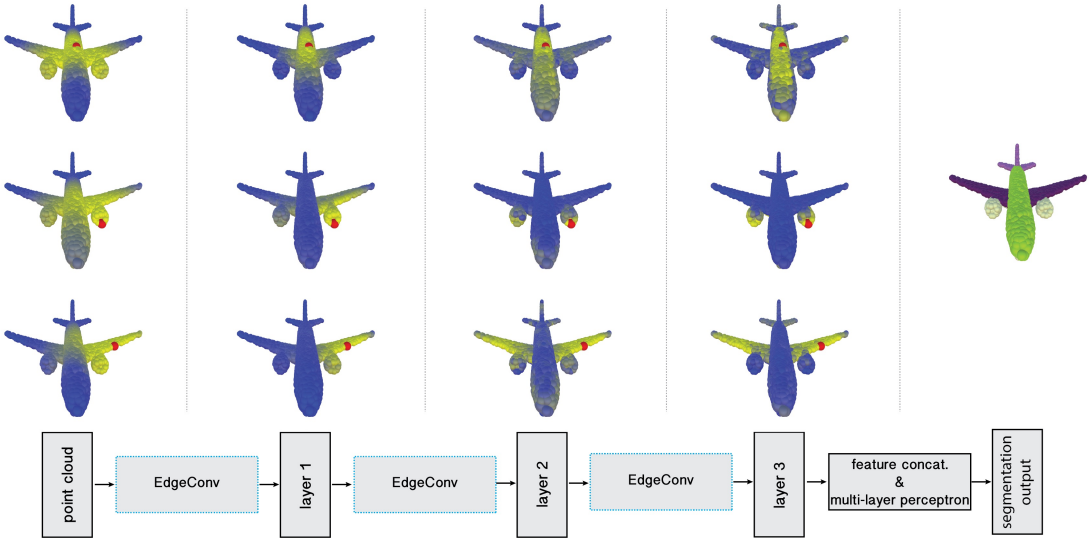


Figure 2.7: **Dynamically recalculating k-nn graphs in the feature space.** DGCNN's distinctive feature is its property of dynamically recalculating the k-nn graph in the learned feature space. The figure illustrates how the k-nn graph of the ● keypoint changes after each application of a dynamic EdgeConv in the current feature space. Points belonging to semantically similar object parts such as wings, turbines and fuselags are located close to each other in deeper feature spaces neglecting the Euclidean distance separating them. The figure is adopted from [WSL+19].

mension. The feature representation of the current vertex i are *updated* by completely forgetting the old state and replacing it with the aggregated incoming messages $\mathcal{A}_i$:

$$\gamma\left(v_i, \mathcal{A}_i; \psi\right) = \mathcal{A}_i \tag{2.7}$$

In conclusion, we yield the following graph convolutional equation for DGCNN:

$$\begin{aligned} v_i' &= \gamma\left(v_i, \square_{j\in\mathcal{N}(i)}\, \phi\left(v_i, v_j, e_{j,i}; \theta\right)\right) \\ &= \max_{j\in\mathcal{N}(i)} \varphi\left([v_i, v_j - v_i]; \theta\right) \end{aligned} \tag{2.8}$$

Dynamic vs static. A unique feature of DGCNN is its property of *dynamically* recalculating neighborhoods and spanning a graph between a vertex and its k nearest neighbors in feature space before applying a subsequent EdgeConv. Thus, we deal with a series of graphs $\left(\mathcal{G}^{(0)}, \ldots, \mathcal{G}^{(l)}, \ldots, \mathcal{G}^{(L)}\right)$ with vertex sets $\mathcal{V}^{(l)}$ iteratively mapped into learned feature spaces $\mathcal{F}^{(l)}$ and dynamically recalculated edge-index sets $\mathcal{I}^{(l)}$ in this very space. This allows to learn non-local features not based on the Euclidean locality of points but their proximity in feature space. In Figure 2.7, the feature space is grouped in specific areas where points belonging to certain object parts are located next to each other while other parts are in distant positions. Since we operate on k-nn graphs, these groups of vertices will intra-connect with vertices in the same group which results in building graph clusters in the feature space, while these clusters get refined with each applied edge convolution.

Contrastingly, edge convolutions can also be performed in a *static* fashion where the graph is not recalculated after applying an edge convolution. Mathematically, we again produce a series of graphs but here, the vertices in the initial vertex set $\mathcal{V}^{(0)}$ are mapped to latent feature spaces $\mathcal{F}^{(l)}$ while the edge-index set $\mathcal{I}$ stays constant throughout all the network. Here, we do not benefit of obtaining graph clusters in latent feature spaces as we do for the dynamic version. However, this dynamic recomputation of edges comes with a critical downside. Computationally, calculating k-nn neighbors lies in $\mathcal{O}(n^2)$ for time and space in the number n of vertices. For arbitrarily large point clouds, this becomes a severe bottleneck and prevents from building deep dynamic graph CNNs. Li *et al*. [LMTG19] show in their work that static but deep graph convolutional networks outperform their dynamic but shallow counterparts.

2.4 Multi-Scale Hierarchies on Meshes

In this chapter, we motivate the importance of multi-scale hierarchies in general and, in particular, for geometric deep learning. Recent publications for 2D and 3D image understanding rely on multi-scale hierarchies [TQD+19, GEvdM18]. A multi-scale hierarchy on meshes consists of a sequence of *hierarchy levels* of the same initial mesh. With each level, the initial mesh is further simplified. Thus, each hierarchy level consists of fewer data points as the previous level. The pooling operations constitute a surjective function mapping N data points from level $l-1$ to $M < N$ points in level l, necessarily decreasing the number of data points by aggregating features of data points pointing to the same representative.

Pooling operations are widely used, mainly because of the following three properties: (1) Pooling the input signal results in a low-pass filtering which filters out small fluctuations in the signal. This makes the neural network invariant to small transformations [GBC16]. (2) (graph) convolutional kernels defined on coarser data representations lead to an increased receptive field which can capture more global features. Therefore, shallower hierarchy levels are responsible for capturing locally restricted fine-grained features and deeper hierarchy levels catch features particularly useful for global localization. (3) Moreover, the reduced number of data points comes with a smaller computational burden for the algorithm. We are thus able to build more complex architectures with fewer data points involved.

Multi-Scale hierarchies in discrete 2D and 3D spaces. Pooling operations initially gained popularity in the field of 2D image understanding. Here, multi-scale hierarchies are generated using permutation-invariant pooling functions (e.g. $\max(\cdot)$, $\min(\cdot)$ and $\text{mean}(\cdot)$) for aggregating features from nearby pixels. We therefore define a cubical pooling kernel which is slid over the input data in a convolution-like fashion. The kernel size f defines the receptive field of the kernel and the striding factor s defines the step size of the convolution. In Figure 2.8, we illustrate max pooling for discrete signals in 2D.

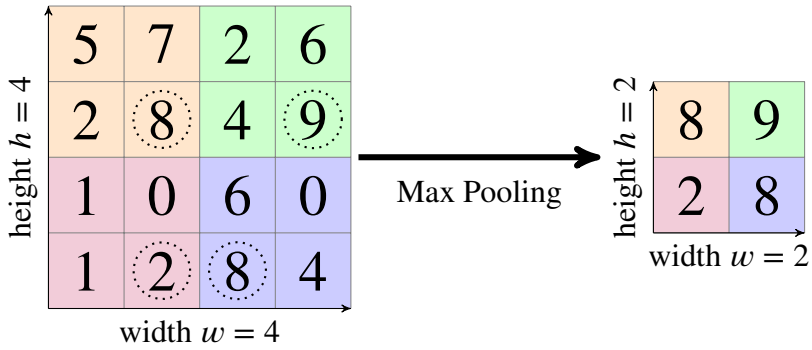


Figure 2.8: **Max Pooling for discrete signals.** We show max pooling with kernel size of $f = 2$ and a stride of $s = 2$ (non-overlapping). Features in the locality of the kernel are aggregated in a permutation-invariant fashion (here: $\max(\cdot)$) to obtain a representative of this neighborhood. We thus significantly reduce the spatial size of the signal resulting in robustness to small transformations as well as increasing the receptive field of subsequent convolutions.

This idea is easily extensible to 3D as well as sparse signals. Here, the pooling kernel is defined in three dimensions and for sparse signals, active sites analogously to sparse convolutions are used [GEvdM18] (see Chapter 2.2.2). An inherent problem of sparse convolutions is solved since pooling operations can merge previously disconnected components that could not be bridged with plain submanifold convolutions. Thus, pooling is a measure to increase the information flow within a point cloud.

Mesh simplification. In this work, we particularly focus on deep learning on graph- and mesh-based data structures. As outlined in the previous chapters, recently, multi-scale architectures gain much popularity due to their performance gains [TQD+19, GEvdM18, CGS19]. Leveraging pooling methods operating directly on discrete and continuous signals (e.g. images and point clouds) fall short since we would not be able to use the graph connectivity information in deeper hierarchy levels. We thus resort to mesh simplification algorithms as the atomic operation for building hierarchy levels on meshes. Analogously to pooling operations, mesh simplification is a natural choice since it reduces the number of vertices, edges and faces while maintaining some user-defined quality requirements which bound the introduced geometric distortion of simplification. Moreover, we have to take care of the topology of the mesh. The question arises if we want manifolds to merge and thus, connecting previously disconnected parts during the simplification process. For example, for medical imaging this is critical but for rendering purposes or semantic segmentation, this poses benefits since we gain a better information flow through the graph neural network [GH97].

In the following of this chapter, we present non-iterative and iterative methods represented by Vertex Clustering [RB93] and Quadric Error Metrics [GH97] as representatives for both groups. We highlight their benefits and downsides and the area where they are best applicable.

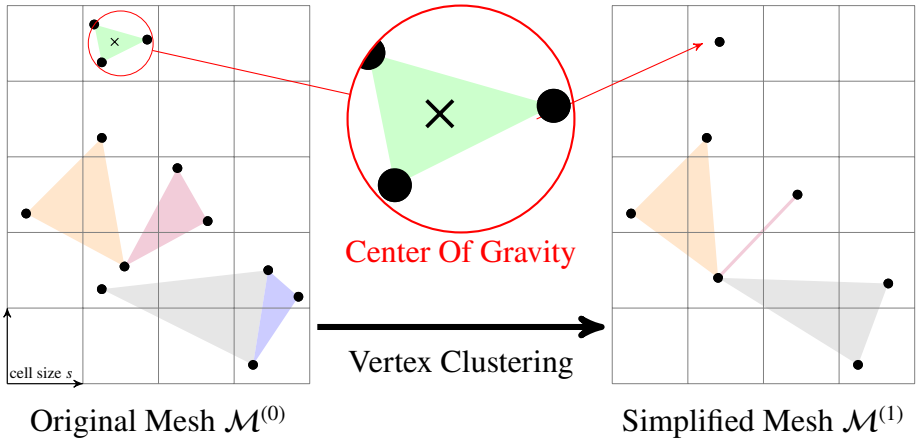


Figure 2.9: **Vertex Clustering.** Using an uniform grid placed on top of the mesh, the new representative vertex is defined as the *center of gravity* of the vertices which fall into the same cell of size s. Connectivity is preserved by keeping track of aggregated vertices to their corresponding occupied grid cells. Degenerated artifacts like *dangling edges*, *disconnected vertices* and drastic topological alterations might occur.

2.4.1 Vertex Clustering

Vertex Clustering as introduced by Rossignac *et al.* [RB93] simplifies the mesh in a non-iterative way without being dependent on the underlying polygonal mesh structure. Therefore, the output does not have to be a triangular mesh and might contain polygons of arbitrary degree and even isolated vertices and edges which are not part of faces.

The fundamental idea of this algorithm is to place a uniform grid with cell size s on top of the mesh and aggregate vertices that fall into the same cell. The new representative for a cell is defined as the *center of gravity* of the contained vertices. The algorithm keeps track of the connectivity of vertices and connects two newly introduced representatives if at least two vertices in their responsible cells have been connected previously. Intra-cell connections are simply deleted since the corresponding vertices have been collapsed together. In Figure 2.9, we present an illustration of a Vertex Clustering step with cell size s and show some important degenerate artifacts which might occur during the process. We highlight that Vertex Clustering is able to make drastic topological changes to the mesh. In this example, the orange, red and gray triangle have been disconnected in the original mesh but share a vertex in the simplified mesh. While allowing better information propagation by connecting disconnected components, this behavior might result in critically altered surface information and hinders the learning process. Moreover, Vertex Clustering can generate artifacts known as *dangling edges* (see the blue edge in the simplified mesh which is not part of any face). In terms of graph neural networks, this however does not pose any problems. Certainly, an *isolated vertex* does. Here, we perform graph convolutions on vertices which neighborhood sets consist only of themselves. Useful feature extraction is not possible anymore.

However, Vertex Clustering provides significant benefits. The cell size parameter $s \in \mathbb{R}_{>0}$ introduces a geometric error bound. The distance between a representative

Figure 2.10: **Quadric Error Metrics.** The cow model is iteratively simplified to models using 5,804, 994, 532, 248 and 64 faces, respectively. The special appeal of using Quadric Error Metrics is their preservation of distinct features, e.g. horns and hooves, throughout many simplification steps. The figure is adopted from [GH97].

and the vertex itself cannot exceed $\sqrt{3} \cdot s$. Thus, Vertex Clustering shares the property of *locality pooling* with standard 2D/3D pooling approaches on 2D images or 3D point clouds. We therefore consider Vertex Clustering as a simplification algorithm which especially takes care of low-pass filtering the vertex density of the mesh.

Challenging for Vertex Clustering is its dependency of the placement and orientation of the surrounding grid which results in drastic changes to the output mesh [GH97]. Moreover, the properties of the simplified mesh are challenging to control. Provided by the cell size parameter s only, we can not control the number of vertices or faces in the resulting mesh nor the maximal allowed geometric error introduced by the simplification. Despite its flaws, Vertex Clustering has gained much popularity, not only due to its simplicity and low runtime.

2.4.2 Quadric Error Metrics

In this chapter, we present Quadric Error Metrics in order to approximate the geometric distortion introduced by contracting pairs of vertices [GH97]. The goal is to *iteratively* simplify the mesh by contracting the pair of vertices which adds the smallest geometric distortion to the simplified model (see Figure 2.10). In each iteration, we choose the pair contraction placed on a heap which introduces the minimal geometric error. Quadric matrices $\mathbf{Q} \in \mathbb{R}^{4\times4}$ heuristically approximate this error in a compact and memory-efficient way. Moreover, it preserves primary and distinct features of the original model throughout all simplified variants.

Contraction of vertex pairs. The distinguishing feature of Quadric Error Metrics is its iterative simplification. The iterative process is achieved by an atomic operator (*contraction of vertex pairs*) which generates a locally simplified mesh. We thus obtain a sequence of simplified meshes $(\boldsymbol{M}_0, \ldots, \boldsymbol{M}_N)$ for N performed atomic vertex pair contractions. If the quality requirements are fulfilled for a particular mesh $\boldsymbol{M}_n$ (e.g. number of faces / vertices, maximal geometric error), we extract the mesh $\boldsymbol{M}_n$ from the sequence and use it a level in our multi-scale hierarchy.

The question arises which vertex pairs (v, w) should be subject to a potential contraction. In the simplest case, we only consider pairs of vertices that are adjacent in the mesh. We thus ensure that every contraction cannot join previously disconnected

components of the mesh. However, this procedure comes with its flaws. Firstly, the simplification process might be to restricted. For instance, a surface (e.g. floor, ceiling) consisting of multiple disconnected parts cannot be merged and we simply end up with deleting parts of the surface due to the constraint of minimizing the number of vertices within the mesh. A better approach constitutes the merging of previously disconnected parts (e.g. introduced by scanning artifacts) to form a connected representation of the entire surface which can then be approximated by a small number of vertices. This behavior is especially desirable when defining a graph neural network over the mesh. Here, we particularly interested in allowing an information flow through previously disconnected parts of the mesh. Disconnected components hinder this propagation of information and might lead to worse segmentation performances.

Therefore, Garland *et al*. [GH97] introduce a threshold t where pairs of non-adjacent vertices (v, w) are treated as valid pairs if their Euclidean distance does not exceed the threshold t: $||v - w||_2 < t$. The parameter t is subject to hyperparameter tuning. For $t = 0$, we only consider vertices connected by edges and for $t \to \infty$, we consider all n^2 pairs of vertices neglecting the distance between them which might result in a significant drop in runtime.

Exact geometric error in terms of distance to original planes. Performing the contraction $(v, w) \to r$ of the pair of vertices (v, w) to a new representative r introduces geometric distortion. $\Delta(r)$ is defined as the squared distance from r to its assigned planes $\mathbf{p}$ which have been assigned to v or w previously, and constitutes a metric in which we measure the costs which are introduced by performing this contraction:

$$\Delta(r) = \sum_{\mathbf{p} \in \text{planes}(r)} \left(\mathbf{p}^T r\right)^2 = \sum_{\mathbf{p} \in \text{planes}(r)} r^T \left(\mathbf{p}\mathbf{p}^T\right) r = r^T \left(\sum_{\mathbf{p} \in \text{planes}(r)} K_{\mathbf{p}} \right) r = r^T \mathbf{Q}_r r \tag{2.9}$$

Here, the vector $\mathbf{p}$ constitutes the normalized normal of the corresponding planes of vertex r. Therefore, the scalar product $\mathbf{p}^T r$ results in the signed distance from the vertex r to plane $\mathbf{p}$.

The *fundamental error quadric* $K_{\mathbf{p}}$ used in the formula $d(x, \mathbf{p}) = x^T K_{\mathbf{p}} x$ returns the squared distance of an arbitrary point x in space to plane $\mathbf{p}$:

$$K_{\mathbf{p}} = \mathbf{p}\mathbf{p}^T = \begin{pmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{pmatrix} \tag{2.10}$$

In the initial mesh, we assign a plane $\mathbf{p}$ to a vertex v if v is a corner of the face embedded in the plane $\mathbf{p}$. Thus, the geometric error for an initial vertex v is simply $\Delta(v) = 0$. When the algorithm contracts the pair (v, w) to the representative r, the union of planes of v and w (e.g. planes(r) = planes$(v) \cup$ planes(w)) will be associated with the representative r in the shape of a list of fundamental error quadrics $K_{\mathbf{p}}$.

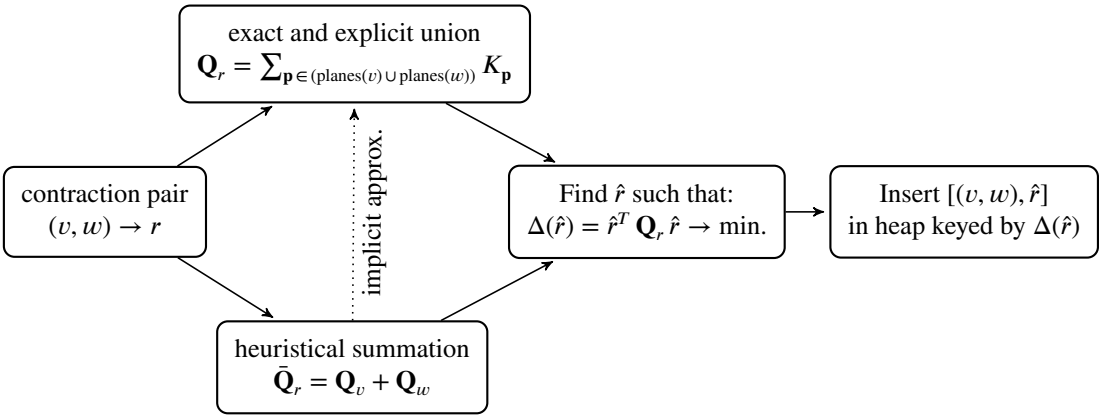


Figure 2.11: **Updating the heap for a contraction pair.** During the initialization phase of the algorithm and when a contraction is performed, we maintain a heap data structure where we store potential contractions keyed by their corresponding contraction costs ascendingly. Instead of explicitly calculating the exact union of original planes associated with a vertex, we resort to an implicit method where we allow overlaps in the aggregation of original planes.

Minimizing $\Delta(r)$ yields the exact position of r. As we see in Equation 2.11, the fundamental error quadric $K_\mathbf{p}$ is a symmetrical 4×4 matrix. The summation of an arbitrary number of fundamental error quadrics thus leads to a symmetrical matrix, as well. Therefore, Equation 2.9 poses a quadratic problem which minimum can be found solving a linear equation system:

$$\begin{pmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} r = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.11}$$

In the case that the matrix is not invertible, Garland *et al.* [GH97] search for the minimum along the edge (v, w). If this fails, they take the minimal cost of the endpoints v and w or their midpoint.

After performing a contraction $(v, w) \to r$, the fundamental error quadrics $K_\mathbf{p}$ of v and w are transferred to r and contraction costs and new representatives for all affected contractions are recalculated. In the exact setting, we therefore have to explicitly recalculate the error quadric $\mathbf{Q}_r$ based on the updated list of fundamental error quadrics. Although exact, this method needs a lot of computations and memory to maintain the heap of valid contraction pairs and to store the list of fundamental error quadric $K_\mathbf{p}$ of the original planes. Thus, this method is computationally unfavorable.

Approximating the geometric error. In contrast to the exact method, Garland *et al.* [GH97] resort to an approximate method which significantly improves the computational time but still guarantees an upper bound for overapproximating the geometric

Algorithm 2: Incremental Decimation with Quadric Error Metrics [GH97].

Input : Initial mesh $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V} \subseteq \mathbb{R}^3$ in 3D space, and edge-index set $\mathcal{E} \subseteq \mathcal{V}^2$. Let $t \geq 0$ be the threshold for selecting pairs.
Output: simplified mesh $\mathcal{G} = \left(\mathcal{V}^{(k)}, \mathcal{I}^{(k)}\right)$ after k steps

Initialize the cost $\Delta(v) = 0$ for each vertex v;
Calculate error quadric $\mathbf{Q}_v$ for each vertex $v \in \mathcal{V}$;
foreach *pair of vertices* $(v, w) \in \mathcal{V}^2$ **do**
  if $(v, w) \in \mathcal{E}$ *or* $||v - w||_2 < t$ **then**
    add (v, w) to the set of valid pairs;
end
foreach *pair of vertices* $(v, w) \in$ *valid pairs* **do**
  if *optimal placement* **then**
    `/* see Equation 2.9 */`
    Find r which minimizes $\Delta(r) = r^T \left(\mathbf{Q}_v + \mathbf{Q}_w\right) r$
  else
    $r \leftarrow \text{argmin}\,(\Delta(v), \Delta(w))$;
  end
  Insert $[(v, w), r]$ on the heap keyed by the contraction cost $\Delta(r)$;
end
while *convergence criterion not met and heap is not empty* **do**
  `/* criterion may be a maximal geometric error or a specified number of vertices or faces */`
  pop top pair (v, w) with replacement vertex r;
  perform contraction to r and delete $[(v, w), r]$ from heap;
  exchange v and w entries in heap with r and recalculate r' and $\Delta(r')$;
end

error. Quadric error matrices $\mathbf{Q}$ (cf. Equation 2.9) constitute a compact and well localized way of keeping track of contraction costs. Rewriting this formula, we end up with a quadratic formulation of $\Delta(r) = r^T \mathbf{Q}_r r$ which allows capturing the error shape with the matrix $\mathbf{Q}_r$. In contrast to *explicitly* calculating $\mathbf{Q}_r$ for a representative $(v, w) \rightarrow r$ based on the union of associated original planes $\mathbf{p}$ in the shape of the sum of fundamental error quadrics $K_\mathbf{p}$, we *implicitly* calculate the sum of the error quadrics $\mathbf{Q}_v$ and $\mathbf{Q}_w$ to obtain an estimate $\bar{\mathbf{Q}}_r = \mathbf{Q}_v + \mathbf{Q}_w$. $\bar{\mathbf{Q}}_r$ shows the same mathematical properties as the exact version since the sum of two symmetrical matrices produces a symmetrical matrix again, as well. Thus, the optimal placement of r is calculated in the same way as in the exact method. A conceptual illustration is depicted in Figure 2.11.

However, this update regime leads to a source of approximation errors. Instead of explicitly taking the union of the planes of v and w, we allow an overlap in the aggregation of planes by simply adding them up. In the case of previously disjoint vertices, the sum of quadric matrices equals the explicitly calculated sum of fundamental error quadrics

$K_{\mathbf{p}}$ over the set of planes $\mathbf{p}$. However, if vertices have been connected previously, we count the corresponding planes multiple times. Since we deal with triangular meshes, an error quadric is added twice as much, tops. Although overapproximating the introduced error, the update rule becomes greatly simplified since it only has to maintain a single 4×4 *symmetric* matrix (10 floating point numbers) instead of an arbitrary large list of fundamental error quadrics $K_{\mathbf{p}}$ of associated original planes $\mathbf{p}$.

Implementation details. We are interested in performing the contractions ordered by the introduced geometric error ascendingly. We, therefore, use a heap data structure keyed by the contraction cost $\Delta(r)$. We store the contraction pair (v, w) and the new representative $\hat{r}$. When we perform a contraction $(v, w) \rightarrow \hat{r}$, we need to find the elements in the heap containing the vertices v and w and update their entries by recalculating the optimal replacement vertex r' with corresponding contraction costs. Furthermore, we delete newly introduced duplicates. In Algorithm 2, we show a pseudo-code implementation of incremental decimation with Quadric Error Metrics.

2.5 Sampling Methods

Unlike discrete spaces, we cannot define an upper limit of points within a certain volume. An explicit representation of an unbounded neighborhood or point cloud is computationally not feasible since we deal with hardware restrictions in terms of memory. Thus, we need to resort to approximate methods. For this purpose, we draw samples from the point cloud or neighborhood distribution to approximate the point cloud or neighborhood with only a fraction of data points. In an efficient manner, sampling methods select a set of representative points from the full-sized neighborhood or point cloud set regarding some user-defined quality measurements (e.g. number of sampled points K or the upper limit of the sample density distribution).

In the following two chapters, we survey sampling methods for sampling the Euclidean space point clouds are embedded into as well as sampling neighborhoods which are particularly interesting for convolutional approaches.

2.5.1 Sampling Continuous Point Clouds

Unlike quantized point clouds (e.g. for enabling discrete submanifold sparse convolution in Chapter 2.2.2), we cannot provide an upper bound for points in the scene. Since we deal with restricted hardware, especially in terms of memory, we need to limit the number of points in the scene. Therefore, sampling becomes inevitable. In the following, we chronologically present prominent sampling methods for continuous point clouds showing their benefits, weaknesses and their computational complexity.

Uniform sampling. Thomas *et al.* [TQD+19] and Simonovsky *et al.* [SK17] use uniform sampling (also called: VoxelGrid sampling) which can be seen as a special case of Vertex Clustering (see Chapter 2.4.1). It discretizes the continuous space by placing a cubical grid over the point cloud and replaces all points which fall into the same cell with a representative point calculated as the center of gravity. While being computationally cheap as its complexity lies in $\mathcal{O}(n)$, this sampling method is invariant to the underlying distribution of the point cloud. Thus, samples in regions characterized by a low density are over-represented in the method's output. Moreover, this sampling method does not provide any randomization to the algorithm. In each iteration, the same samples are drawn which might lead to overfitting.

Random sampling. PointNet [QLJ+17] leverages random sampling which selects K points from the point cloud randomly. Although being computationally cheap and eliminating uniform sampling's downside of being invariant to the point distribution, this method under-represents areas in the scene which have substantially less points. Especially when considering LiDAR point clouds, this downside becomes apparent. Objects placed nearby to a LiDAR sensor have significantly more samples than distant objects. However, the fixed sample size of K samples is particularly useful to harness the parallelization capabilities of today's GPUs.

Farthest point sampling (FPS). Qi *et al.* [QYSG17] address the downsides of random sampling while still guaranteeing a fixed sample size K. Farthest Point Sampling (FPS) iteratively adds points to the set of sampled points if their distance to the rest of the set is maximal. Compared with random sampling, they claim that this results in better coverage of the scene since also points of less populated regions are likely to be drawn. However, this comes with a computational overhead of $\mathcal{O}(n^2)$ since in each run of FPS, all distances between points have to be recalculated.

Inverse density sampling (IDS). Groh *et al.* [GWL18] propose a method which adapts FPS such that it still provides good coverage of the scene while being significantly faster. They approximate the density at a specific point x in space as follows:

$$\widetilde{\rho}(x) = \sum_{x' \in \mathcal{N}(x)} ||x - x'||_2 \tag{2.12}$$

The neighborhood $\mathcal{N}(x)$ can be obtained using a k-nn or radius search. Samples are then drawn proportional to the unnormalized density distribution $\widetilde{\rho}$. IDS benefits from the fact that the density estimates of points only have to be calculated once. For later training epochs, new samples can be easily drawn from the estimated density distribution. Thus, sampling becomes a $\mathcal{O}(n)$ problem in the number n of points in the point cloud. Moreover, they claim that this approach is useful for convolutional approaches since they have already calculated point neighborhoods in a preprocessing step.

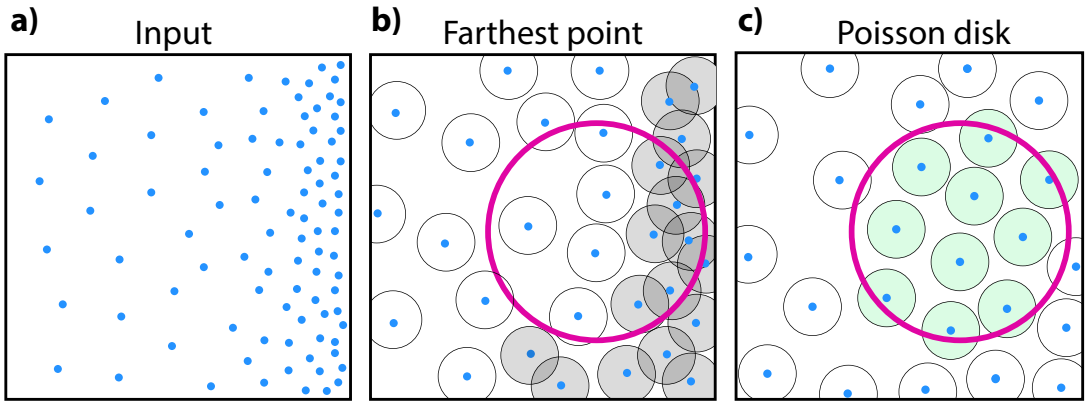


Figure 2.12: **Bounding the density by Poisson disk sampling.** By defining a maximal number of non-overlapping Poisson disks per unit sphere, we introduce an upper bound of disks placed in a certain region. We therefore limit the bias of a sample being drawn from a high-density region as well as limiting the maximal number of neighbors by the Kepler conjecture [HAB+17]. Contrastingly, we see that FPS is not capable of restricting the maximal number of points being drawn from a high-density region. This figure is adopted from [HRV+18].

Poisson disk sampling (PDS). Similar to IDS, Poisson disk sampling [HRV+18] aims to draw samples according to the point cloud distribution while emphasizing sparser over dense regions. However, it has its special appeal of guaranteeing an upper bound of points per unit sphere, thus being particularly well-suited for convolutional approaches (see Figure 2.12). The idea of PDS is to replace each drawn point sample by a sphere with a certain radius r_p (called: Poisson disk) which is not allowed to overlap with other spheres. Using the Kepler conjecture [HAB+17], this introduces an upper bound n of Poisson disks which might occur in a sphere with radius $r \geq r_p$:

$$n < \frac{\pi \left(r + \frac{r_p}{2} \right)^3}{3\sqrt{2} {r_p}^3} \tag{2.13}$$

However, this does not result in a fixed number K of drawn samples. PDS is therefore not able to leverage the acceleration methods used by the GPU to reserve memory of constant sizes in advance.

2.5.2 Sampling Continuous Neighborhoods

Convolutional approaches learn locally restricted features dependent on their neighborhood. Wang *et al.* [WSL+19] and Engelmann *et al.* [EKL20] leverage k-nn graphs to approximate the neighborhood for convolutions. k-nn graphs consist of the k nearest neighbors to a specific center point given an arbitrary distance measure. k-nn graphs are beneficial since they always return the same number of neighboring points and adapt to regions with diverse densities naturally.

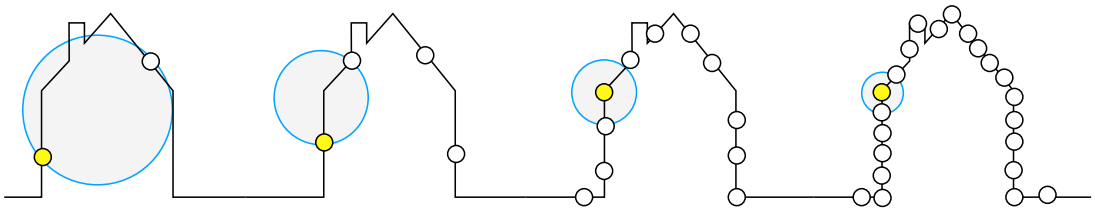


Figure 2.13: **k-nn approaches under varying sampling densities.** The receptive field of a $k = 2$-nn neighborhoods is significantly smaller in dense sampling settings and sparser in low density settings. Thus, k-nn approaches are particularly vulnerable to sampling approaches which do not preserve the overall density distribution of the point cloud in contrast to approaches relying on radius neighborhood. This figure is adopted from [HRV+18].

However, Hermosilla *et al.* [HRV+18] and Thomas *et al.* [TQD+19] argue that k-nn graph approaches suffer from non-uniform densities in the point cloud (see Figure 2.13). Thus, they propose to use radius graphs to define the notion of neighborhoods for points. However, very densely populated regions can lead to arbitrarily large neighborhoods, which introduces a computational burden for convolutional methods. Sampling the neighborhood space becomes inevitable.

Approaches like MCCN [HRV+18] and KPConv [TQD+19] control the density of the point cloud by sampling it via uniform or Poisson disk sampling which both guarantee an upper bound of samples within a unit sphere. However, this falls short since the challenge is not the number of samples drawn from the distribution; it is the size of the neighborhood which is prohibitive for convolutions. Therefore, the concurrent work of Lei *et al.* [LAM19] randomly samples the neighborhood to obtain at most K samples for approximating the neighborhood set while leaving the actual point cloud unchanged.

In this work, we propose *Random Edge Sampling* (RES) which is similar in spirit to the work of Lei *et al.* [LAM19] but its distinctive feature is the probabilistic interpretation of reducing the expected size of the neighborhood set (see Chapter 3.3).

3 DualConvNet

The purpose of this thesis is to prove that a combination of geodesic and Euclidean convolutions improves the feature representation per vertex. With this in mind, we introduce a novel family of deep hierarchical network architectures called *DualConvNet*. Its goal is to leverage a simple but well-established architecture for semantic segmentation in order to ensure modularity and measurability of all components. Thus, we are able to base our claims about geodesic and Euclidean convolutions on a variety of experiments with each component observed individually (see Chapter 5). The idea of the architecture is to combine geodesic and Euclidean convolutions in *dual convolution modules* such that the network explores the 3D surface mesh and the Euclidean domain simultaneously in each step of the algorithm.

When applying geodesic convolutions on the mesh, we explicitly learn features focusing on the surface structure of the scene. Its distinctive feature is that it neglects geodesically remote but spatially close vertices. Hence, we assume that geodesic convolutions are more likely to learn feature representations for object shapes. Contrastingly, Euclidean convolutions focus on the Euclidean proximity of vertices in terms of k-nn or radius neighborhoods in 3D space. They enable an information flow between geodesically disconnected parts of the scene and therefore, we assume that they learn the interaction between objects.

Modern architectures leverage multi-scale hierarchies in order to learn feature representations at different resolutions from fine-grained, highly localized features to coarse but semantically enriched features of larger patches. Geodesic convolutions however rely on a preserved mesh structure at each hierarchy level in the network. Thus, we describe the necessary mesh-centric pooling operations, such as our extensions to Vertex Clustering and Quadric Error Metrics by introducing *pooling trace maps*. Pooling trace maps allow to convert high-resolution to low-resolution meshes and vice versa while guaranteeing user-defined quality requirements of the mesh structure (see Chapter 2.4).

Since Quadric Error Metrics does not define an upper bound of the vertex density in the continuous Euclidean domain, we cannot guarantee a maximal number of neighbors that we have to consider for applying Euclidean convolution on radius graphs. Thus, it is inevitable to resort to sampling methods to approximate the radius neighborhood for

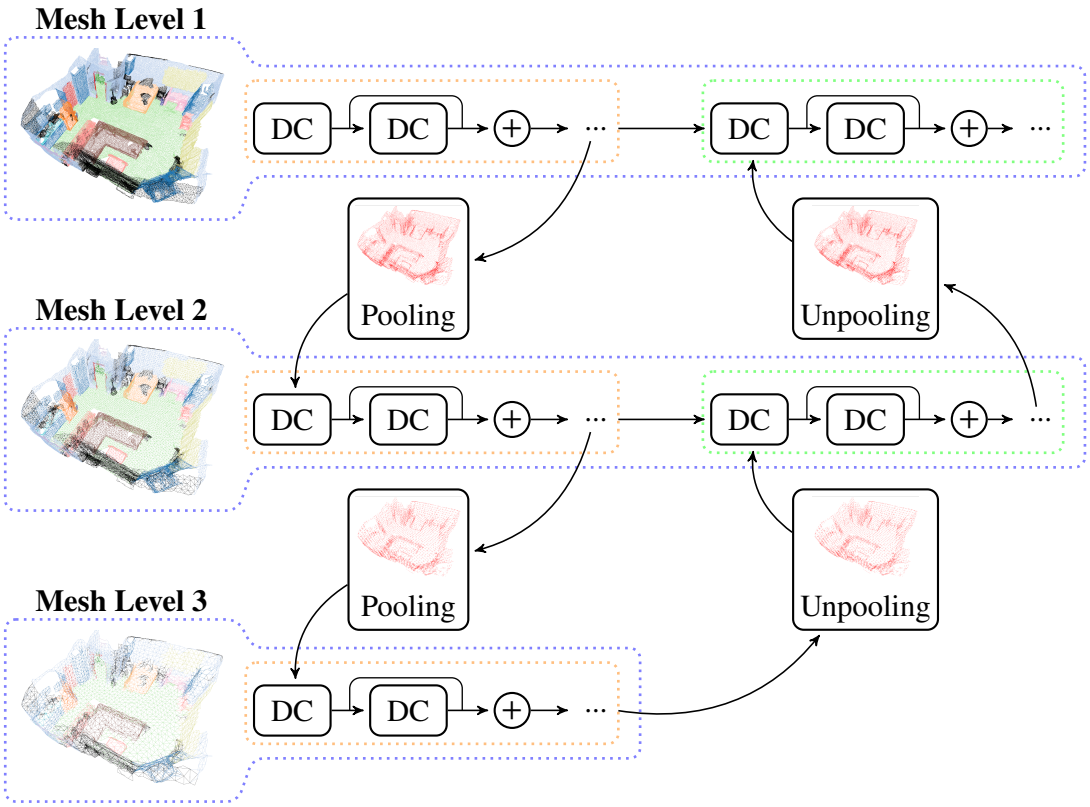


Figure 3.1: **Architecture of DualConvNets.** DualConvNets follow the symmetrical encoder-decoder architecture of U-Nets [RFB15]. Several dual convolutions (DC) comprising Euclidean and geodesic convolutions are stacked to enlarge the receptive field in both the spatial domain as well as on the 3D mesh surface structure. They are bypassed by skip connections for better convergence in each graph level. (Un)pooling operations with pooling trace maps generated by mesh simplification algorithms are leveraged to convert the mesh to different resolutions for feature learning at various scales.

enabling convolutions. We therefore introduce Random Edge Sampling (RES). RES is a probabilistic sampling method that reduces the potentially unlimited neighborhood to an expected sampling size. Thus, allowing us the vary the size during training as well as test, and produce predictions based on a better approximation of the neighborhood.

3.1 Network Architecture

In Figure 3.1, we present our multi-scale DualConvNet architecture which adopts the U-Net architecture [RFB15] as its fundamental building block. We follow the same symmetrical encoder-decoder architecture with additional skip-connections interconnected them in the same hierarchy level. At each hierarchy level, we perform several dual convolutions (see Figure 3.2) in a consecutive manner. Dual convolutions com-

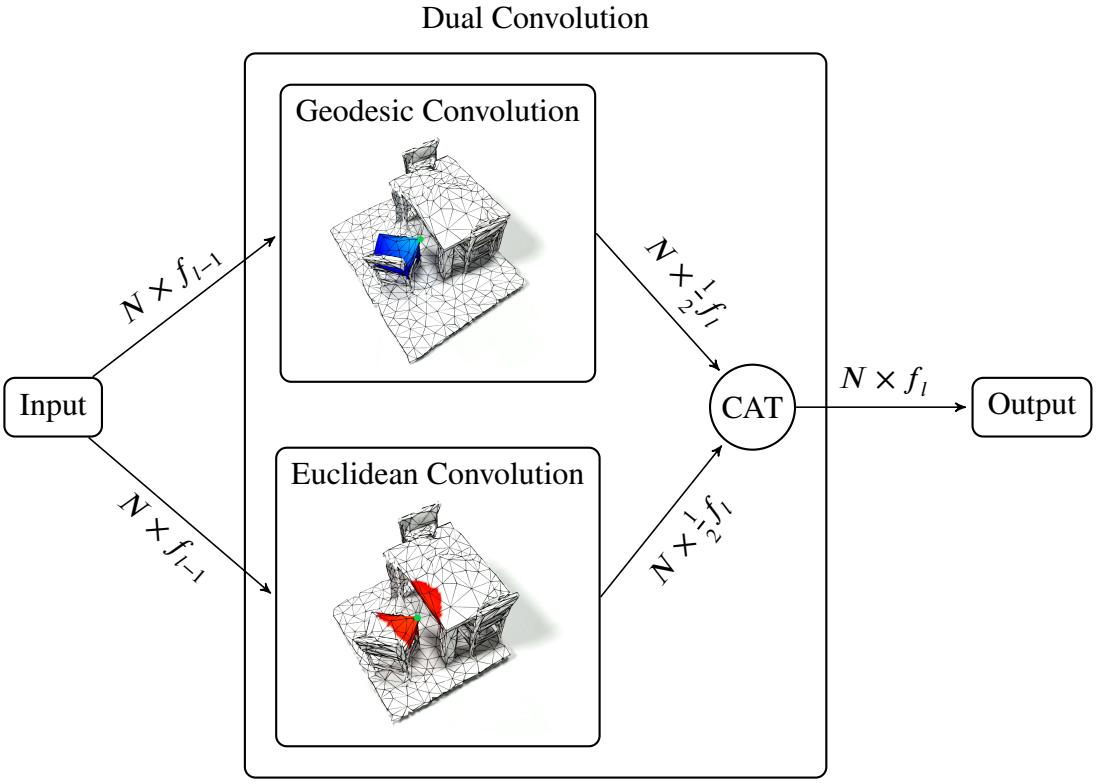


Figure 3.2: **Dual Convolution Module.** Geodesic and Euclidean convolutions are applied in a parallel manner. Thus, we enlarge the receptive field in the spatial domain as well as on the surface mesh structure simultaneously. Input features of size $N \times f_{l-1}$ of N vertices with f_{l-1} features are independently processed by geodesic and Euclidean convolutions which output the halved output feature size in order to be comparable with SingleConvNet architectures.

prise geodesic and Euclidean convolutions applied in parallel. Thus, we can learn features in both the Euclidean domain as well as on the 3D surface mesh by enlarging the receptive field in both domains simultaneously. The resulting features of both convolutions are concatenated and serve as input for the following dual convolution. In order to enable a better convergence, we by-pass each dual convolution module with skip-connections to allow an unhindered gradient flow as proposed by He *et al*. [HZRS16]. We learn features at various scales in order to capture fine-grained, highly localized details at high-resolution hierarchy levels and coarse but semantically enriched features for large patches in low-resolution representations of the same mesh. We thus define the necessary (un-)pooling operations relying on pooling trace maps generated from mesh simplification algorithms.

We use a U-Net like architecture due to its modularisation capabilities. In the ablation study, we individually measure the impact of all architectural components. Here, we vary the mesh simplification algorithm for generating low-resolution mesh representationx, as well as the number of hierarchy levels used by the architecture. Moreover, we

vary the number of dual convolutions and even disable them altogether to measure the effect of each convolution independently and in parallel. Hence, we make reasonable claims about the combination of geodesic and Euclidean convolutions.

The DualConvNet family makes comparison between geodesic and Euclidean convolutions simple. We will refer to an instantiation of our network that only operates in a single space as a *SingleConvNet*, whereas our full model operating simultaneously in both spaces is referred to as a *DualConvNet*. Note that SingleConvNet are a subset of the family of DualConvNets since they equal DualConvNets if we set the number of filters of the second convolution type to 0 everywhere.

3.2 Euclidean and Geodesic Graph Convolutions

Dual convolutions comprise Euclidean and geodesic convolutions. As we have presented in Chapter 2.2, there exists a vast variety of convolutions for the 3D and graph space including convolutions applied on discrete and continuous as well as sparse and dense settings. Whereas the decision to use graph convolutional neural networks for learning geodesic features on the 3D surface mesh lies at hand, the decision which convolution type to use for Euclidean convolutions is more involved. We use graph convolutions for both domains in order to limit the number of hyperparameters to our algorithm and to ease the comparison of the results. In this work, we focus on the neighborhood definitions which differentiate *geodesic* convolutions from *Euclidean* ones (see Figure 1.2): *Geodesic graph convolutions* use 1-hop neighborhoods, i.e., all vertices which are connected to the center vertex by exactly one edge. Therefore, this defines a locally defined surface patch of the mesh. In parallel, *Euclidean graph convolutions* rely on the Euclidean neighborhood of a vertex i which is defined over the Euclidean distance between pairs of vertices. In this work, we use Euclidean neighborhoods limited in their size by using k-nn or radius graphs. We thus ensure Euclidean locality which is an essential property of convolutions (see Chapter 2.2.1). We compare both approaches in Chapter 5.

Convolution operator. We perform graph convolutions on the graph $G^\ell = (V^\ell, E^\ell)$ induced by the underlying mesh M^ℓ of hierarchy level ℓ. Note that $E^\ell = E_g^\ell \cup E_e^\ell$ is the (not necessarily disjunctive) union of the geodesic edge set E_g^ℓ, induced by the faces of M^ℓ, and the Euclidean edge set E_e^ℓ, obtained from the k-nn or radius graph neighborhood of each vertex $i \in V^\ell$. We implement convolutional layers over vertex features v_i associated with vertex i similar to *EdgeConvs* [WSL+19]. Specifically, the output features $v_i' \in \mathbb{R}^E$ of vertex i with input feature $v_i \in \mathbb{R}^F$ are computed as

$$v_i' = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \varphi([v_i, v_j - v_i]; \theta) \tag{3.1}$$

where $\mathcal{N}_i$ is the geodesic/Euclidean neighborhood of the vertex i, $|\mathcal{N}_i|$ its cardinality, φ is implemented as an MLP with trainable parameters θ and $[\cdot, \cdot]$ is the concatenation.

Contrarily to the original implementation of EdgeConvs [WSL+19] (Chapter 2.3.2), we adapted the definition slightly: instead of max(·) aggregating features per vertex, we calculate the mean of the receiving messages $\varphi([v_i, v_j - v_i]; \theta)$ from neighboring nodes j. In contrast to DGCNN [WSL+19], we do not restrict ourselves to k-nn neighborhoods but allow radius neighborhoods. Here, we cannot guarantee a pre-defined neighborhood size which leads to a high variance of features v_i' when taking max features of neighborhoods with varying cardinalities. Hence, we resort to averaging over all receiving features which smooths incoming features. Thus, our convolution is robust against a varying size of the neighborhood.

Note that the number of kernel parameters θ is independent of the kernel size induced by the neighborhood $\mathcal{N}_i$. This is in contrast to 2D CNNs, where the number of parameters increases quadratically with the kernel size.

In contrast to DGCNN [WSL+19], we do not recalculate the neighborhoods in the learned feature space but we reuse the 3D positions of vertices for the Euclidean and geodesic domain. Skipping this dynamic recalculation of neighbors allows us to create deeper graph convolutional networks, which turned out to be more effective [LMTG19].

Translation equivariant convolutions. On the very first convolution in the network, we define a translation equivariant version of the dual convolution module which does not rely on absolute positions. Providing absolute positions to our algorithm leads to learning that specific objects tend to appear at specific positions in the scene. For instance, if the training set contains many chairs at a specific position, this prior distribution biases the network to predict chair features in that region during test time, even if no chairs occur in this area. Therefore, we apply $\varphi(\cdot)$ to $v_j - v_i$ and do not concatenate the initial features containing absolute positions (cf. Equation 3.1).

3.3 Random Edge Sampling (RES)

Hermosilla *et al.* [HRV+18] argue that radius neighborhoods increase the robustness to non-uniformly sampled point clouds in contrast to k-nn ones. Since the simplification with Quadric Error Metrics does not guarantee uniformly sampled meshes in the Euclidean domain, we rely on radius neighborhoods. However, radius neighborhoods may lead to an arbitrarily large number of neighbors. Convolving over these unbounded neighborhoods, leads to a considerably large computational load which we wish to reduce. We thus resort to sampling methods to approximate the neighborhood.

As a technical contribution, we define a novel sampling method on graph neighborhoods, called *Random Edge Sampling (RES)*. Motivated by Srivastava *et al.* [SHK+14], we randomly sample edges from the Euclidean edge set E_e^ℓ on all hierarchy levels ℓ. By doing so, we introduce more variety in the training data and thus increase the generalization of our approach, while simultaneously reducing the computational load.

To implement RES, we define a function $D : \mathcal{N}_i \to [0, 1]$ which maps the neighborhood $\mathcal{N}_i = \{v_1, \dots, v_{|N_i|}\}$ of a given vertex v_i to its corresponding sampling probability.

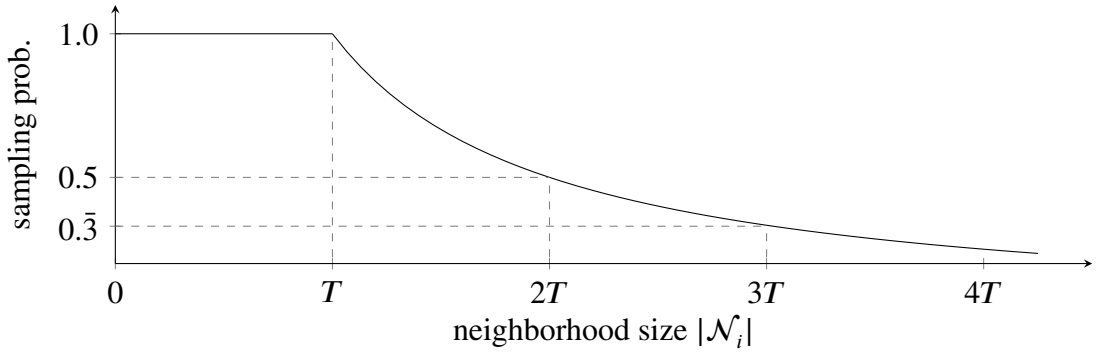


Figure 3.3: **Sampling probabilities for Random Edge Sampling (RES)**. We only sample neighborhoods with a size greater than T in order not to further reduce already small sets. Edges in neighborhoods exceeding the threshold T are sampled by a probability computed by $D(\mathcal{N}_i)$, resulting in $\mathbb{E}[|\mathcal{N}_i|] \leq T$.

Each edge established between v_i and $v \in \mathcal{N}_i$ is subsequently sampled with the same probability $D(\mathcal{N}_i)$. D is defined as follows:

$$D(\mathcal{N}_i) = \begin{cases} 1 & \text{if } |\mathcal{N}_i| \leq T \\ T \cdot |\mathcal{N}_i|^{-1} & \text{if } |\mathcal{N}_i| > T \end{cases} \tag{3.2}$$

We introduce a threshold T such that small neighborhoods are not furtherly decimated. However, edges in sets exceeding the threshold T undergo sampling. After applying RES, the expected size of the neighborhood is bounded by $\mathbb{E}[|\mathcal{N}_i|] \leq T$.

$$\begin{aligned} \mathbb{E}[|\mathcal{N}_i|] &= \sum_{j \in \mathcal{N}_i} D(\mathcal{N}_i) \cdot 1 = D(\mathcal{N}_i) \cdot \sum_{j \in \mathcal{N}_i} 1 \\ &= D(\mathcal{N}_i) \cdot |\mathcal{N}_i| \overset{(3.2)}{\leq} T \cdot |\mathcal{N}_i|^{-1} \cdot |\mathcal{N}_i| \leq T \end{aligned} \tag{3.3}$$

We visualize the function $D(\mathcal{N}_i)$ in Figure 3.3. Varying the threshold T equals to varying the expected number of edges we draw from the neighborhood set. We experience that decreasing the threshold during training leads to a good approximation of the neighborhood while saving computational resources. During inference time, we increase the sample size using larger neighborhood sample sizes for obtaining final predictions. We conducted ablation studies for RES in Chapter 5.

3.4 Pooling using Mesh Simplification

In this work, we investigate the potential of combining geodesic and Euclidean convolutions. It has been shown to be beneficial to learn feature representations on multiple scales in a scene [GEvdM18, TQD+19, RFB15] because finer representations allow to learn highly localized features whereas coarser representations capture the context better. However, in contrast to Euclidean convolutions, it is not sufficient for multi-scale

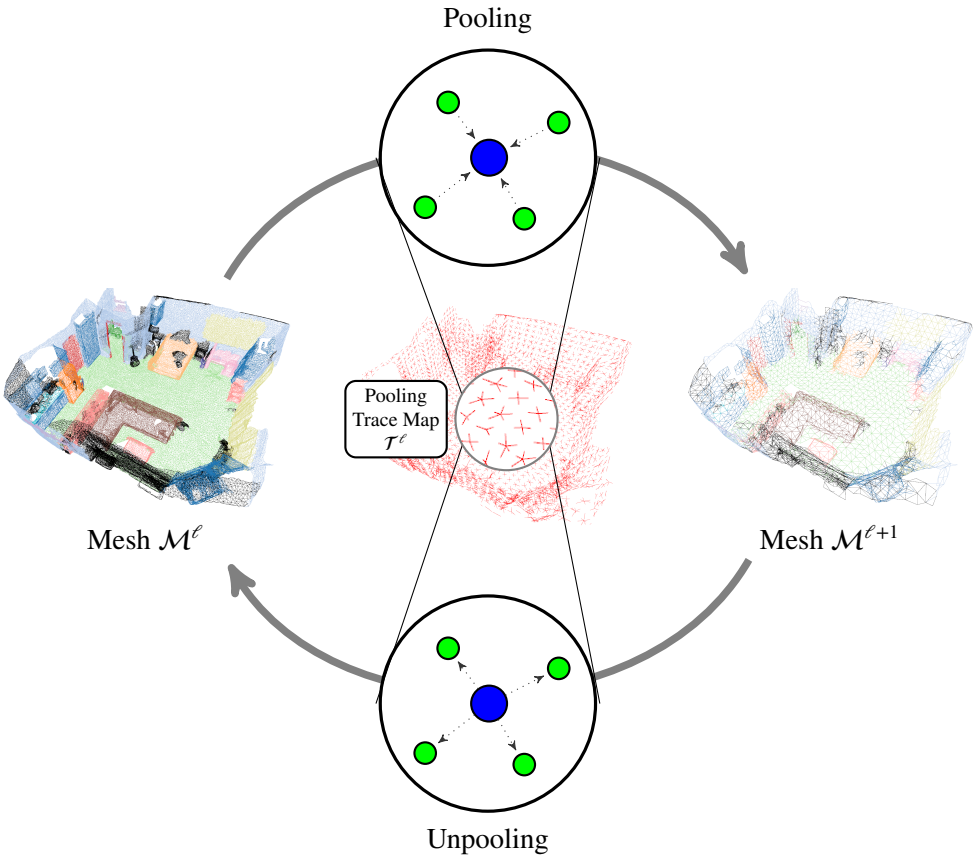


Figure 3.4: **Mesh Simplification as (un-)pooling operators.** We need to preserve the surface information on all multi-resolution hierarchy levels in order to apply geodesic convolutions. We therefore use mesh simplification algorithms such as vertex clustering and quadric error metrics as (un-)pooling operators. We implement them as simple look-up dictionaries which we call *pooling trace maps* (shown in red). They store vertex connectivities between original vertices and their corresponding representatives throughout the mesh hierarchy. We perform pooling with permutation-invariant aggregation functions and unpooling by simply copying features of the representative vertex to its corresponding vertices of the previous level.

geodesic feature extractors to downsample the point cloud without preserving edge information of the mesh. We take special provisions to preserve the essential surface information in subsequent hierarchy levels for applying geodesic convolutions. Thus, we define pooling operators which generate and interconnect these hierarchy levels for learning feature representations on multiple mesh scales. These operators should fulfill the following two requirements: (1) Pooling operations should reduce the number of vertices while retaining the overall surface information of the original mesh. (2) Pooling operations are bijective functions from partitions of the vertex set $\mathcal{V}^\ell$ of level ℓ to representative single vertices of the vertex set $\mathcal{V}^{\ell+1}$ of the next hierarchy level.

Pooling trace maps for connecting hierarchy levels. We leverage mesh simplification algorithms for generating hierarchies of meshes $(\mathcal{M}^0, \dots, \mathcal{M}^\ell, \dots, \mathcal{M}^\mathcal{L})$ connected by *pooling trace maps* $(\mathcal{T}^0, \dots, \mathcal{T}^\ell, \dots, \mathcal{T}^{\mathcal{L}-1})$ (see Figure 3.4). We consecutively perform mesh simplification steps (e.g. Vertex Clustering or Quadric Error Metrics) on the initial mesh $\mathcal{M}^0$ which represents the mesh at its finest resolution and its successors until we obtain $\mathcal{M}^\mathcal{L}$ which is the coarsest representation of the same mesh after the final mesh simplification step. We extend mesh simplification algorithms such that they generate not only new mesh representatives but also provide look-up pooling maps which interconnect vertices of predecessing levels to their new representative vertices in consecutive hierarchy levels. Therefore, a pooling trace map $\mathcal{T}^\ell$ maps a partition of vertices $\{v_i^{\ell-1}\} \subset \mathcal{V}^{\ell-1}$ bijectively to a single representative vertex $v^\ell \in \mathbb{R}^3$ in the next graph level ℓ, which is again connected with vertices within the same graph level via the edge set E_g^ℓ obtained by the mesh simplification algorithm. Since the pooling trace map maps partitions of vertices to single representative vertices, we conclude that $|\mathcal{V}^{\ell-1}| \leq |\mathcal{V}^\ell|$. Thus, we fulfill requirement (1). Moreover, considering this bijection, we can *trace* initial vertices with their corresponding representatives throughout the complete multi-scale hierarchy. We therefore establish a one-to-one relationship of vertex features of each graph level with their corresponding initial vertices. Notably, there exist no intermediate vertices in the hierarchy which are not part of any traces and all traces are complete, e.g. they reach the final hierarchy level. Similar to Pan *et al*. [HSYX18], we apply a permutation invariant aggregation function (e.g., sum(·), max(·) or mean(·)) on the features of $\{v_i^\ell\}$ to obtain pooled features for $v^{\ell+1}$. Unpooling is performed by merely copying features of representative vertices to their corresponding vertices of the predecessing hierarchy level.

Mesh simplification as (un-)pooling. We have presented Vertex Clustering (VC) (Chapter 2.4.1) and Quadric Error Metrics (QEM) (Chapter 2.4.2) as two mesh simplification algorithms from the geometry processing domain. These algorithms reduce the number of vertices, preserve the overall surface information and introduce minimal geometric error. We extend them with pooling trace maps to obtain (un-)pooling in our architecture through simple look-up operations.

We modify the Vertex Clustering approach as follows: vertices that fall into the same grid cell of length s are contracted to a new representative vertex $v^{\ell+1}$ of the next hierarchy level. We store the mapping between the representative vertex $v^{\ell+1}$ and its corresponding vertices $\{v_i^\ell\}$ of the previous level in the pooling trace map.

Alternatively, we consider Quadric Error Metrics. In contrast to VC, this approach incrementally contracts vertex pairs (v, w) to a new representative r according to an approximate error of the geometric distortion this contraction introduces. We keep track of performed contractions. When the algorithm reaches a specific quality property (e.g. number of vertices or faces, introduced geometric error), we create a snapshot of the current mesh which we insert into the mesh hierarchy. Since intermediate representatives r might be subject to contractions, we condense the list of performed contractions to a list only considering initial vertices to final representatives of the current snapshot.

Since there is a 1-to-1 relationship between contraction pairs and their corresponding representatives, this task essentially represents a graph search linearly in the number of initial vertices. As VC aims for uniform vertex density and QEM for minimal geometric distortion, we compare both orthogonal approaches in our ablation study in Chapter 5.

4 Experiments

In this chapter, we put the focus on experimentally evaluating our proposed architecture. After discussing the standard evaluation metrics, we present qualitative and quantitative results on three well-established 3D scene datasets. Among graph convolutional approaches, we report new state-of-the-art results on all datasets. We provide detailed descriptions on the training and testing pipeline, including data preprocessing, data augmentation as well as network architectures used to obtain the final scores.

4.1 Datasets

To prove the validity of our proposed method, we evaluate it on three well-established 3D scene datasets. We follow the standard evaluation protocol of each benchmark in order to provide comparable results. Moreover, we conduct an ablation study to analyse the influence of each component individually.

Stanford Large-Scale 3D Indoor Spaces (S3DIS) [ASZ+16]. S3DIS contains 3D point clouds from 6 large-scale indoor areas obtained by high-resolution scans from Matterport cameras, consisting of 271 rooms from 3 different university buildings. Among others, room types include offices, lecture halls, lavatories, hallways as well as open spaces. The RGB point cloud is annotated with 13 semantic classes in a per-point fashion. Our algorithm depends on meshes as its input data structure. However, S3DIS only includes 3D meshes which are not semantically annotated. As the resolution of these meshes is low compared to the point cloud resolution, we oversample all faces and interpolate the color and ground truth information from the semantically annotated ground truth points. To obtain comparable results on the official benchmark, our final predictions are backpropagated to the original point cloud. More information about the S3DIS preprocessing is given in Setion 4.3. We follow the same evaluation protocol as proposed by Armeni *et al.* [ASZ+16] in order to be comparable with other approaches. In the first setting, we train on all areas except Area 5, which we keep for testing. In the second setting, we provide k-fold cross-validation results over all areas.

ScanNet v2 Benchmark [DCS+17]. Unlike S3DIS, the ScanNet v2 benchmark provides 3D meshed point clouds containing normal, RGB color information as well as semantic labels on a per-vertex basis. Similarly to S3DIS, ScanNet comprises scans obtained with high-resolution Matterport cameras from a wide variety of indoor rooms. They used 20 valid semantic classes in contrast to S3DIS's 13 classes. ScanNet established a publicly available online benchmark including hidden test labels[1].

We perform all our experiments using the official training, validation, and test split of 1201, 312, and 100 scans, respectively. We perform the ablation study in Chapter 5 on the ScanNet validation set in order to show the effect of all components.

Matterport3D [CDF+17]. The most recent dataset we consider is Matterport3D which contains meshed reconstructions of 90 reconstructions of building-scale RGBD scans. In contrast to ScanNet's room-wise paradigm, Matterport3D scans are captured in a building-wise manner which allows reconstructing whole buildings. For the sake of comparison, we stick to the same evaluation protocol as introduced in 3DMV [DN18], and TextureNet [HZY+19] and report mean recall scores on 21 classes of the publicly available test set.

4.2 Evaluation Metrics

In order to quantitatively evaluate the performance of a model, we define quality metrics. Here, we give scores measuring how well the algorithm predicts each class individually. We thus define two distributions: The ground truth class distribution $\hat{Y}(x, c)$ and the predicted class distribution $Y(x, c)$ which both yield 1 if the data point x belongs to (/ is classified as) class c. In order to make quantitative statements about an algorithm, we compare both distributions. *Evaluation metrics* are the means to perform this comparison which is presented in the following paragraphs. The notation is adopted from Zhao *et al.* [ZGWmC19].

Atomic evaluation metrics. In order to quantitatively reason about how well an algorithm performs a semantic segmentation task, we define atomic evaluation metrics. Each of them focuses on an orthogonal aspect of evaluating the quality of a prediction individually. Combining these atomic metrics to more elaborate ones leads to conclusive quality metrics paying attention to various aspects.

The number of *true positives (TP)* constitutes how many data points of the ground truth class c were correctly classified:

$$\mathrm{TP}(Y, \hat{Y}, c) = \sum_{x} Y(x, c) \cdot \hat{Y}(x, c) \tag{4.1}$$

[1] The public leaderboard is available under:
http://kaldir.vc.in.tum.de/scannet_benchmark/

The number of *true negatives (TN)* comprises how many data points not belonging to the ground truth class c were correctly rejected by the predictor:

$$\mathrm{TN}(Y, \hat{Y}, c) = \sum_x (1 - Y(x, c)) \cdot \left(1 - \hat{Y}(x, c)\right) \tag{4.2}$$

The number of *false positives (FP)* represents how many data points not belonging to the ground truth class c were wrongly classified by the predictor:

$$\mathrm{FP}(Y, \hat{Y}, c) = \sum_x Y(x, c) \cdot \left(1 - \hat{Y}(x, c)\right) \tag{4.3}$$

The final metric is the number of *false negatives (FN)* which represents how many data points of the ground truth class c were wrongly rejected by the predictor:

$$\mathrm{FN}(Y, \hat{Y}, c) = \sum_x (1 - Y(x, c)) \cdot \hat{Y}(x, c) \tag{4.4}$$

As the previously presented quality metrics constitute absolute numbers of correctly and wrongly classified data points, we cannot compare performances across various datasets. We thus define condensed metrics defined in a range of $[0, 1]$ for enabling comparison on a relative scale. One metric we are interested in is *precision (Prec)* which measures the ratio between true positives and the overall number of class instantiations. Precision returns a probability estimate of how *exact* our predictor is:

$$\mathrm{Prec}(Y, \hat{Y}, c) = \frac{\mathrm{TP}(Y, \hat{Y}, c)}{\mathrm{TP}(Y, \hat{Y}, c) + \mathrm{FP}(Y, \hat{Y}, c)} \tag{4.5}$$

Besides precision, there is the metric of *recall (Rec)* which is defined as the fraction of true positives to the overall positive samples. This estimates how *complete* a prediction is:

$$\mathrm{Rec}(Y, \hat{Y}, c) = \frac{\mathrm{TP}(Y, \hat{Y}, c)}{\mathrm{TP}(Y, \hat{Y}, c) + \mathrm{FN}(Y, \hat{Y}, c)} \tag{4.6}$$

The compromise between precision and recall. The precision rate contains information about how *exact* a prediction is. A high precision rate therefore states that positive predictions are likely to be classified correctly. Contrastingly, the recall rate measures how *complete* a prediction is. Hence, a high recall rate means that instantiations of a specific class are likely to be detected in the dataset.

We illustrate both metrics in the following examples considering an autonomously driving vehicle: High recall rates are particularly useful for safety-related driving functions. It is of utmost importance that our algorithm does not miss out on pedestrians crossing the street in order to avoid fatal accidents. If, by mistake, the algorithm detects a pedestrian even if there is none, it might influence the requirement of a comfortable drive but does not harm any safety-critical requirements. However, an exact prediction is crucial for the detection of right-of-way signs. Thus, we need a high precision rate.

This example illustrates that both precision and recall are important metrics to evaluate the performance of a predictor. However, we favor a scalar value to create an ordered ranking of algorithms performing on the same task. We thus condense both metrics into a single one. The F_β measure weights precision and recall according to a weighting factor β such that recall has β times more weight than precision:

$$F_\beta(Y, \hat{Y}, c) = (1 + \beta^2) \frac{\text{Prec}(Y, \hat{Y}, c) \cdot \text{Rec}(Y, \hat{Y}, c)}{\left(\beta^2 \cdot \text{Prec}(Y, \hat{Y}, c)\right) + \text{Rec}(Y, \hat{Y}, c)} \tag{4.7}$$

In the case of $\beta = 1$, the F_1 score is the harmonic mean between precision and recall.

Positively correlated to F_β is the metric of *intersection over union* (IoU) which is prominent in the field of computer vision and is used in the evaluation protocols across all tested datasets of this thesis.

$$\text{IoU}(Y, \hat{Y}, c) = \frac{\text{TP}(Y, \hat{Y}, c)}{\text{TP}(Y, \hat{Y}, c) + \text{FP}(Y, \hat{Y}, c) + \text{FN}(Y, \hat{Y}, c)} \tag{4.8}$$

In order to ease comparison even more, the IoU scores for each class c are further averaged to yield the mean IoU score (mIoU):

$$\text{mIoU}(Y, \hat{Y}) = \sum_c \text{IoU}(Y, \hat{Y}, c) \tag{4.9}$$

4.3 Implementation and Training Details

In this chapter, we will give detailed information on which models we derive from our proposed family of DualConvNets. Moreover, we discuss the training and testing pipeline, including precomputation and augmentation steps. Our models are implemented in PyTorch (Geometric) [FL19, PGC+17] and trained on a Tesla V100 16GB for at least 100 epochs ($\sim$ 8 days). We refer to Chapter 5 for detailed ablation studies on the architectural components and focus on comparison to current state-of-the-art methods in this chapter.

Precomputing the graph hierarchy. Using Vertex Clustering and Quadric Error Metrics as pooling operations, we precompute the hierarchy levels $\mathcal{M}^\ell$ and the corresponding pooling trace maps $\mathcal{T}^\ell$. The cell sizes of VC are set to $4\,\text{cm}^3$, $8\,\text{cm}^3$, $16\,\text{cm}^3$, and $32\,\text{cm}^3$, respectively, for each hierarchy level. However, we perform minor adaptions to QEM. We observe that directly applying QEM on the mesh leads to a high vertex density in noisy areas. These areas are mostly unlabeled for the task of semantic segmentation. Therefore, we first apply VC on the initial mesh with a cell size of $4\,\text{cm}^3$, before applying QEM, where we keep 30% of all vertices in each mesh level.

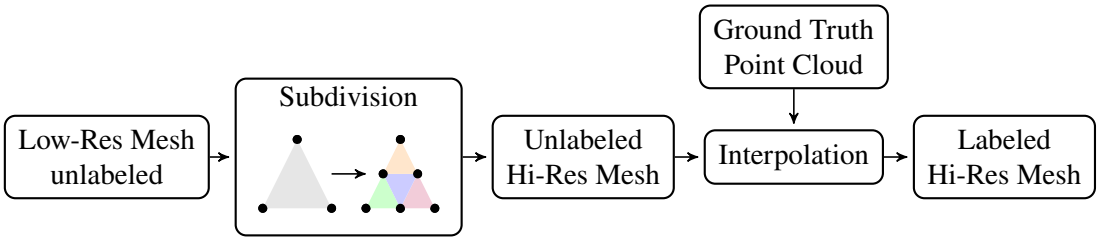


Figure 4.1: **Preprocessing Pipeline for S3DIS.** Our approach requires meshes as input for which the S3DIS data set does not provide an RGB + Label format. Therefore, we establish a preprocessing pipeline in order to leverage low-resolution meshes given by the dataset. Here, we perform midpoint subdivision to artificially enhance the resolution of the mesh, before interpolating RGB colors as well as labels from the ground truth point cloud onto the mesh.

Special provisions for S3DIS. The S3DIS dataset provides high-resolution point cloud data including RGB color information and semantic labels. Moreover, it contains mesh data with significantly lower resolution than the meshes from ScanNet or Matterport3D. In order to use the semantic labels of the official point clouds sampled from these meshes, we artificially enhance the resolution of the low-resolution mesh by splitting edges precisely in the middle if the edge length does not fall under 2 cm under this process. Thus, we create new mesh triangles by connecting the old vertices with their adjacent vertices in the midst of the edges (known as *midpoint subdivision* illustrated in Figure 4.1). Thus, we obtain 4 smaller triangles from the original triangle. By finding the nearest neighbor in the ground truth point cloud for each vertex in the enhanced mesh, we subsequently interpolate the ground truth information, including RGB color information and semantic labels onto this newly created mesh. Note that in contrast to the original point cloud, we thus leverage normal information of the mesh.

Network architectures. In our U-Net like architecture, we extract features at each hierarchy level by three dual convolutions, including ReLU activation functions as well as batch normalization (see Figure 3.1). Even if we deal with small batch sizes, e.g. 4, we achieve better results using traditional batch normalization [IS15] than with incorporating the recently proposed group normalization [WH18] which has been particularly designed for such cases.

We observe that altering the ratio between Euclidean and geodesic filters in the dual convolution modules with respect to the simplification degree of a mesh level leads to improved segmentation results. In shallow hierarchy levels, the mesh surface is only slightly simplified. Thus, we can leverage high-frequency signals of the object surfaces by learning more geodesic filters than Euclidean ones. As the resolution of the mesh is further decimated, localizing objects becomes more important. Here, we need more Euclidean than geodesic features. Therefore, we use 75% geodesic filters in the first two hierarchy levels and 25% geodesic filters in the last two levels. A detailed ablation study is given in Chapter 5.2.

In Table 7.4, we show the detailed network architecture for the ScanNet and Matterport3D benchmark. We do not use the same network definition for the S3DIS dataset. Since the original resolution of the S3DIS meshes is low, we do not benefit from increasing the number of geodesic filters in the early levels. Thus, we set the ratio of geodesic convolutions in each level to 50%. In Table 7.3, we provide the adapted network structure.

Neighborhood notion. We experiment with different notions for neighborhoods. When considering the k-nn approach, we use $k = 15$ nearest neighbors for training and testing. Contrastingly, for radius neighborhood we use radius sizes of 12.5cm, 25cm, 50cm and 100cm for each mesh level as the vertex density decreases gradually. We conduct our experiments with random edge sampling with threshold $T = 15$ while training and $T = 25$ while testing, as we observe that a lower threshold for training still leads to good approximations of the neighborhood. Since we use a random sampling method for neighborhoods, the predictions vary in each run. We therefore run each evaluation 10 times and provide mean and standard deviations in our ablation study. A detailed ablation study about random edge sampling is provided in Chapter 5.1.

Rejecting training examples. It is a common practice among recent approaches to discard training samples of low quality. Methods only differ in the quality criteria [QYSG17, QLJ+17]. We reject training examples which have more than 80% unlabeled vertices, which effectively corresponds to 0.8% of the 18,530 cropped training samples of the ScanNet v2 train set. Moreover, if a crop of a scene has less than 50 semantically labeled points, we also reject it because its expressive power is restricted. Analogously, Qi *et al.* [QSMG17] reject training examples if the number of points in a crop falls below a certain threshold and [QYSG17] rejects blocks which number of unlabeled points exceeds a threshold of 70%. Note that we do not apply this method on the validation or test set.

Optimizing the model. We train the network end-to-end by minimizing the cross-entropy loss using the Adam optimizer [KB15] with an initial learning rate of 10^{-3} and step-wise exponential learning rate decay of 0.5 after every 40 epochs with a batch size of 4. We use a weighted version of the cross-entropy loss function in order to cope with class imbalance problems, e.g. floor, walls and ceiling classes tend to outnumber smaller classes such as bathtubs and shower curtains constantly across all datasets:

$$L(\hat{y}, y) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} w_c \cdot \hat{y}_{i,c} \cdot \log y_{i,c} \tag{4.10}$$

Here, $\hat{y}$ represents the one-hot encoded ground truth vector which is set to 1 if the class c is assigned to vertex i. $y_{i,c}$ is the prediction vector constrained by $\sum_c y_{i,c} = 1$ to resemble a probability distribution. The weight vector $w \in \mathbb{R}^C_{\geq 0}$ assigns a weight to each class c such that less frequent classes contribute more to the loss than frequent ones. For the

ScanNet dataset, we stick to the class weights proposed by TextureNet [HZY+19]. For Matterport3D and S3DIS, we calculate the negative logarithm of the class frequency f_c of the train set to put more emphasis on underrepresented classes.

$$w_c = -\log f_c \tag{4.11}$$

Data Augmentation. In order to prevent overfitting of our model, we perform data augmentation on the input meshes. In order to train more complex models, we crop 3m×3m blocks with a stride of 1.5m from the ground plane. For ensuring valid meshes, we delete edges pointing to vertices not contained in the current crop. Moreover, using the pooling trace map, vertices might be pooled to representatives not present in the current block. In this case, we redirect the trace to the nearest neighbor in the following mesh level in order to guarantee a bijective pooling function.

We further process these crops by normalizing positions to the range of $[-1, 1]^3$ and by mapping the RGB color value from range $[0, 255]^3$ to $[0, 1]^3$. Crops are then transformed by a random linear transformation comprising a dedicated rotation component and a random linear transformation, slightly altering the identity matrix to obtain random instantiations of rotation, shearing and scaling. We draw random variables from a Gaussian distribution centered around 0 with ϵ mean as well as the rotation parameter θ drawn from the uniform distribution of $[0, 2\pi]$ in order to create arbitrary rotations around the z-axis.

$$\begin{aligned} a, \dots, i &\sim \mathcal{N}[0, \epsilon] \\ \theta &\sim \text{unif}[0, 2\pi] \end{aligned} \tag{4.12}$$

$$v' = \underbrace{\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{random rotation around } z\text{-axis}} \cdot \underbrace{\begin{pmatrix} 1+a & b & c \\ d & 1+e & f \\ g & h & 1+i \end{pmatrix}}_{\text{random linear transformation}} \cdot v \tag{4.13}$$

Although training on crops, we highlight that our method is translation-equivariant since it does not consider absolute vertex positions (see Chapter 3.2). We are therefore able to perform inference on full scans.

4.4 Results

In this chapter, we provide quantitative and qualitative results on three well-established 3D semantic scene segmentation datasets.

Performance grouped by methods' convolution type. Since most competing approaches evaluate on ScanNet as well as S3DIS Area 5, we provide a condensed overview about the research field in Table 4.1. Here, we show the performance of our approach compared to recent competing approaches grouped by the approaches'

Method	mIoU ScanNet	mIoU S3DIS	Convolution Type
DeepGCN [LMTG19]	-	52.5	Graph Convolutions
SPGraph [LS18]	-	58.0	
SPH3D-GCN* [LAM19]	61.0	59.5	
HPEIN [JZL+19]	61.8	61.9	
DualConvNet **(Ours)**	**65.3**	**63.8**	
PointNet [QSMG17]	-	41.1	Permutation Invariant Networks
PointNet++ [QYSG17]	33.9	-	
FCPN [DJJ+18]	44.7	-	
3DMV [DN18]	48.3	-	2D-3D
JPBNet [CLLH19]	63.4	-	
MVPNet [JGS19]	64.1	62.4	
TangentConv [TPKZ18]	43.8	52.6	Surface Convolutions
SurfaceConvPF* [HSYX18]	44.2	-	
TextureNet [HZY+19]	56.6	-	
PointCNN [LBS+18]	45.8	57.3	Point Convolutions
ParamConv [WSM+18]	-	58.3	
MCCN [HRV+18]	63.3	-	
PointConv [WQL19]	66.6	-	
KPConv [TQD+19]	68.4	**67.1**	
SparseConvNet [GEvdM18]	72.5	-	Voxelized Sparse Convolutions
MinkowskiNet [CGS19]	**73.4**	65.3	

Table 4.1: **Comparison to state-of-the-art.** Semantic segmentation mIoU scores on the offical ScanNet benchmark [DCS+17] and S3DIS Area-5 [ASZ+16]. We clearly outperform other graph convolutional approaches on all benchmarks. * indicates concurrent work. Full network definitions are given in the appendix.

inherent convolution type. We can report state-of-the-art results for graph convolutional approaches by a significant margin of 3.5% mIoU for the ScanNet benchmark, as well as 1.9% mIoU for S3DIS Area 5. Only 4 approaches report better results on ScanNet. These algorithms propose orthogonal improvements by leveraging point convolutions or voxelized sparse convolutions that could be combined with our DualConvNets (see Chapter 6 for an outlook on future work). Moreover, SparseConvNet [GEvdM18] and MinkowskiNet [CGS19] use Voxelized Sparse Convolutions, which currently perform best on ScanNet, but which are inherently limited for other tasks in that they cannot make use of detailed surface information. Figure 4.2 shows qualitative results on the ScanNet validation set.

Matterport3D. We evaluate our algorithm on the Matterport3D dataset [CDF+17] and report overall state-of-the-art results on that benchmark in Table 4.2. We follow

Method	mRec	wall	floor	cab	bed	chair	sofa	table	door	wind	shf	pic	cntr	desk	curt	ceil	fridg	show	toil	sink	bath	other
PointNet++ [QYSG17]	43.8	80.1	81.3	34.1	71.8	59.7	63.5	**58.1**	49.6	28.7	1.1	34.3	10.1	0.0	68.8	79.3	0.0	29.0	70.4	29.4	62.1	8.5
SplatNet [SJS+18]	26.7	**90.8**	**95.7**	30.3	19.9	**77.6**	36.9	19.8	33.6	15.8	15.7	0.0	0.0	0.0	12.3	75.7	0.0	0.0	10.6	4.1	20.3	1.7
TangentConv [TPKZ18]	46.8	56.0	87.7	41.5	73.6	60.7	69.3	38.1	55.0	30.7	33.9	50.6	38.5	19.7	48.0	45.1	22.6	35.9	50.7	49.3	56.4	16.6
3DMV [DN18]	56.1	79.6	95.5	59.7	82.3	70.5	73.3	48.5	64.3	55.7	8.3	55.4	34.8	2.4	**80.1**	**94.8**	4.7	54.0	71.1	47.5	76.7	19.9
TextureNet [HZY+19]	63.0	63.6	91.3	47.6	82.4	66.5	64.5	45.5	69.4	60.9	30.5	**77.0**	**42.3**	44.3	75.2	92.3	49.1	**66.0**	80.1	**60.6**	86.4	27.5
DualConvNet (**Ours**)	**65.6**	77.8	93.5	**65.2**	**88.9**	69.4	**84.7**	44.8	**81.5**	**62.9**	**41.0**	72.8	39.6	**47.8**	58.7	89.2	**66.8**	41.0	**85.5**	49.4	**87.4**	**30.0**

Table 4.2: **Mean recall scores on Matterport3D Test [CDF+17].** We outperform other approaches in 11 out of 21 classes. In Table 7.4, we provide the network definition used for this experiment. Note that we provide mean recall scores which competing methods sometimes call mean class accuracy.

Method	mIoU	mRec	ceil.	floor	wall	beam	col.	wind.	door	chair	table	book.	sofa	board	clut.
Pointnet [QSMG17]	41.1	49.0	88.8	97.3	69.8	0.1	3.9	46.3	10.8	52.6	58.9	40.3	5.9	26.4	33.2
SegCloud [TCA+17]	48.9	57.4	90.1	96.1	69.9	0.0	18.4	38.4	23.1	75.9	70.4	58.4	40.9	13.0	41.6
Eff 3D Conv [ZLU18]	51.8	68.3	79.8	93.9	69.0	0.2	28.3	38.5	48.3	71.1	73.6	48.7	59.2	29.3	33.1
RSNet [HWN18]	51.9	59.4	93.3	98.4	79.2	0.0	15.8	45.4	50.1	65.5	67.9	22.5	52.5	41.0	43.6
TangentConv [TPKZ18]	52.6	62.2	90.5	97.7	74.0	0.0	20.7	39.0	31.3	69.4	77.5	38.5	57.3	48.8	39.8
PointCNN [LBS+18]	57.3	63.9	92.3	98.2	79.4	0.0	17.6	22.8	62.1	80.6	74.4	66.7	31.7	62.1	56.7
RNN Fusion [YLH+18]	57.3	63.9	92.3	**98.2**	79.4	0.0	17.6	22.8	62.1	74.4	80.6	31.7	66.7	62.1	56.7
ParamConv [WSM+18]	58.3	67.1	92.3	96.2	75.9	**0.3**	6.0	**69.5**	63.5	66.9	65.6	47.3	68.9	59.1	46.2
MinkowskiNet [CGS19]	65.4	71.7	91.8	98.7	86.2	0.0	34.1	48.9	62.4	89.8	81.6	74.9	47.2	74.4	58.6
KPConv [TQD+19]	**67.1**	**72.8**	**92.8**	97.3	**82.4**	0.0	23.9	58.0	69.0	**91.0**	81.5	**75.3**	**75.4**	**66.7**	**58.9**
SPGraph [LS18]	58.0	66.5	89.4	96.9	78.1	0.0	**42.8**	48.9	61.6	84.7	75.4	69.8	52.6	2.1	52.2
SPH3D-GCN* [LAM19]	59.5	65.9	93.3	97.1	81.1	0.0	33.2	45.8	43.8	79.7	86.9	33.2	71.5	54.1	53.7
HPEIN [JZL+19]	61.9	68.3	91.5	98.2	81.4	0.0	23.3	65.3	40.0	75.5	**87.7**	58.5	67.8	65.6	49.4
DualConvNet (**Ours**)	**63.8**	71.2	91.4	95.8	78.6	0.0	21.7	61.3	54.8	88.1	78.1	72.1	67.0	66.4	54.2

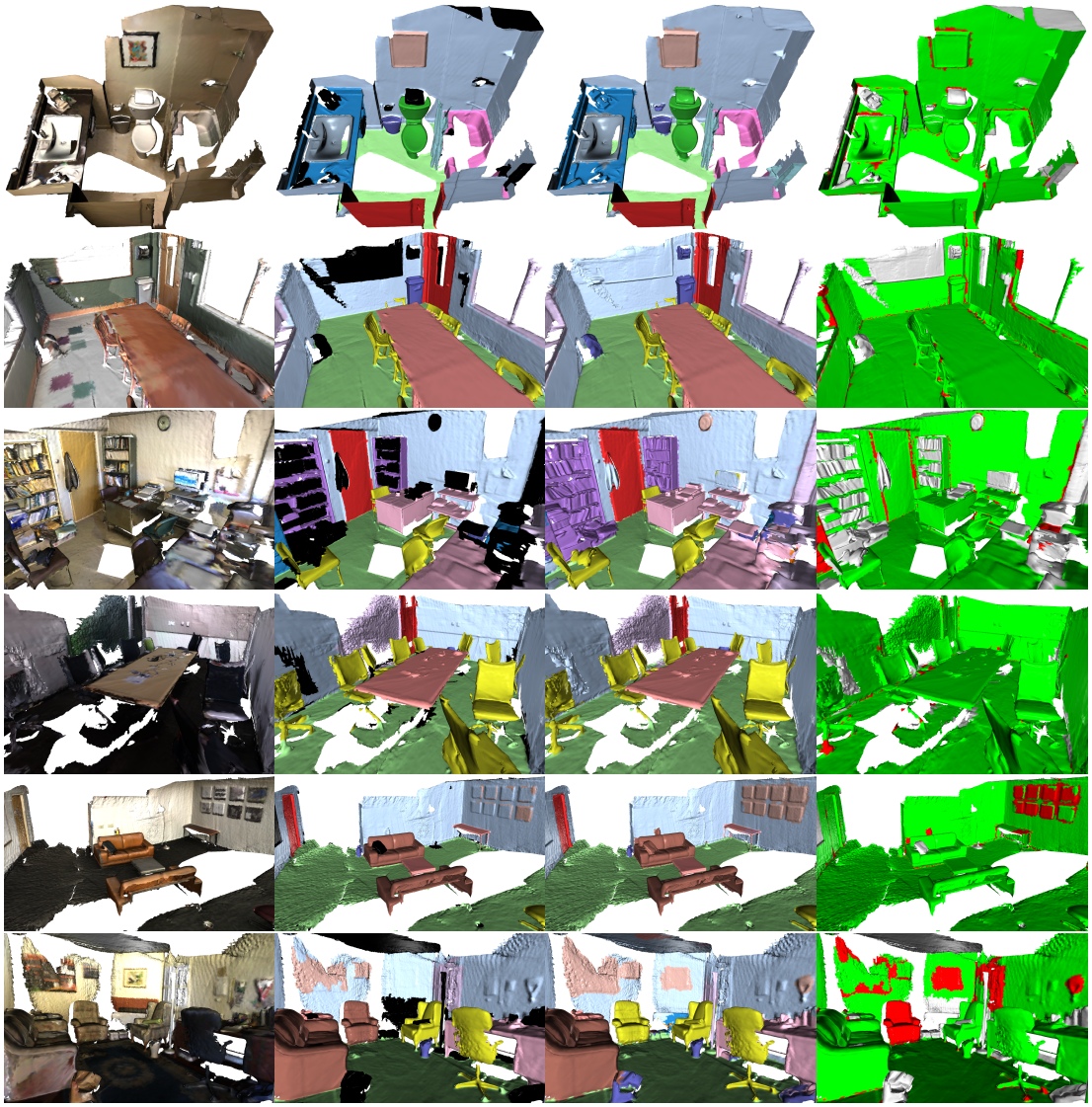
Table 4.3: **Semantic segmentation results on S3DIS Area 5.** We provide mIoU and mean recall scores. Among all approaches, we perform third best only outperformed by KPConv [TQD+19] and MinkowskiNet [CGS19]. Among graph convolutional approaches, we clearly report state-of-the-art with a gap of 1.9% to HPEIN [JZL+19]. The network used for this experiment is given in Table 7.3. Note that we provide mean recall scores which some other competing approaches call mean class accuracy.

the same evaluation protocol as 3DMV [DN18] and TextureNet [HWN18] by only reporting the class-wise recall in contrast to the usually used IoU score. Figure 4.3 shows qualitative results on the Matterport3D test set.

Stanford Large-Scale 3D Indoor Spaces. We furthermore provide class-wise semantic segmentation scores on S3DIS Area 5 (Table 4.3) as well as S3DIS k-fold (Table 4.4). Figure 4.4 shows qualitative results on S3DIS Area 5.

Method	mIoU	mRec	ceil.	floor	wall	beam	col.	wind.	door	chair	table	book.	sofa	board	clut.
Pointnet [QSMG17]	47.6	66.2	88.0	88.7	69.3	42.4	23.1	47.5	51.6	42.0	54.1	38.2	9.6	29.4	35.2
RSNet [HWN18]	56.5	66.5	92.5	92.8	78.6	32.8	34.4	51.6	68.1	60.1	59.7	50.2	16.4	44.9	52.0
PointCNN [LBS+18]	65.4	75.6	**94.8**	**97.3**	75.8	63.3	51.7	58.4	57.2	71.6	69.1	39.1	61.2	52.2	58.6
KPConv [TQD+19]	**70.6**	79.1	93.6	92.4	**83.1**	**63.9**	54.3	66.1	**76.6**	57.8	64.0	**69.3**	**74.9**	61.3	60.3
SPGraph [LS18]	62.1	73.0	89.9	95.1	76.4	62.8	47.1	55.3	68.4	73.5	69.2	63.2	45.9	8.7	52.9
HPEIN [JZL+19]	67.8	76.3	-	-	-	-	-	-	-	-	-	-	-	-	-
SPH3D-GCN* [LAM19]	68.9	77.9	93.3	96.2	81.9	58.6	**55.9**	55.9	71.7	72.1	**82.4**	48.5	64.5	54.8	60.4
DualConvNet (**Ours**)	**69.4**	**80.8**	93.6	96.5	81.0	44.5	44.6	**71.7**	73.8	**74.7**	71.0	63.5	62.5	**62.6**	**62.3**

Table 4.4: **Semantic segmentation results on S3DIS k-fold.** We provide mIoU and mean recall scores. Among all approaches, we perform second best only outperformed by KPConv [TQD+19]. Among graph convolutional approaches, we report state-of-the-art with a gap of 0.5% to the concurrent work SPH3D-GCN [LAM19]. The network definition is given in Table 7.3. Note that we provide mean recall scores which some other competing approaches call mean class accuracy.



Input Mesh **Ground Truth** **Prediction** **Error**

● unlabeled ● wall ● floor ● cabinet ● bed ● chair ● sofa ● table ● door ● window ● bookshelf ● picture ● counter ● desk ● curtain ● fridge ● shower curtain ● toilet ● sink ● bathtub ● otherfurn

Figure 4.2: **Results on ScanNet v2 validation [DCS+17].** Our method correctly predicts challenging classes such as ● image and ● shower curtain, while maintaining clear boundaries even in noisy areas (see ● window in Row 4 and ● picture in Row 6). In the last row, our method correctly predicts ● arm chair even though the ground truth is falsely labeled as ● sofa. Additionally, our algorithm segments ● picture on the ● wall more reliably than the ground truth (see the last two rows). There are some reasonable mistakes like the ● table in Row 5 partly labeled as ● desk.

Figure 4.3: **Results on Matterport3D [CDF+17].** Our method correctly predicts even unlabeled regions such as extending the wall in Row 2, predicting the chimney as other furniture and the whirlpool as bathtub. However, reasonable errors occur, such as confusing windows extending down to the floor as doors and confusing countertops, tables and desks. In the third row, our algorithm correctly predicts sofa even though the ground truth is falsely labeled as chair.

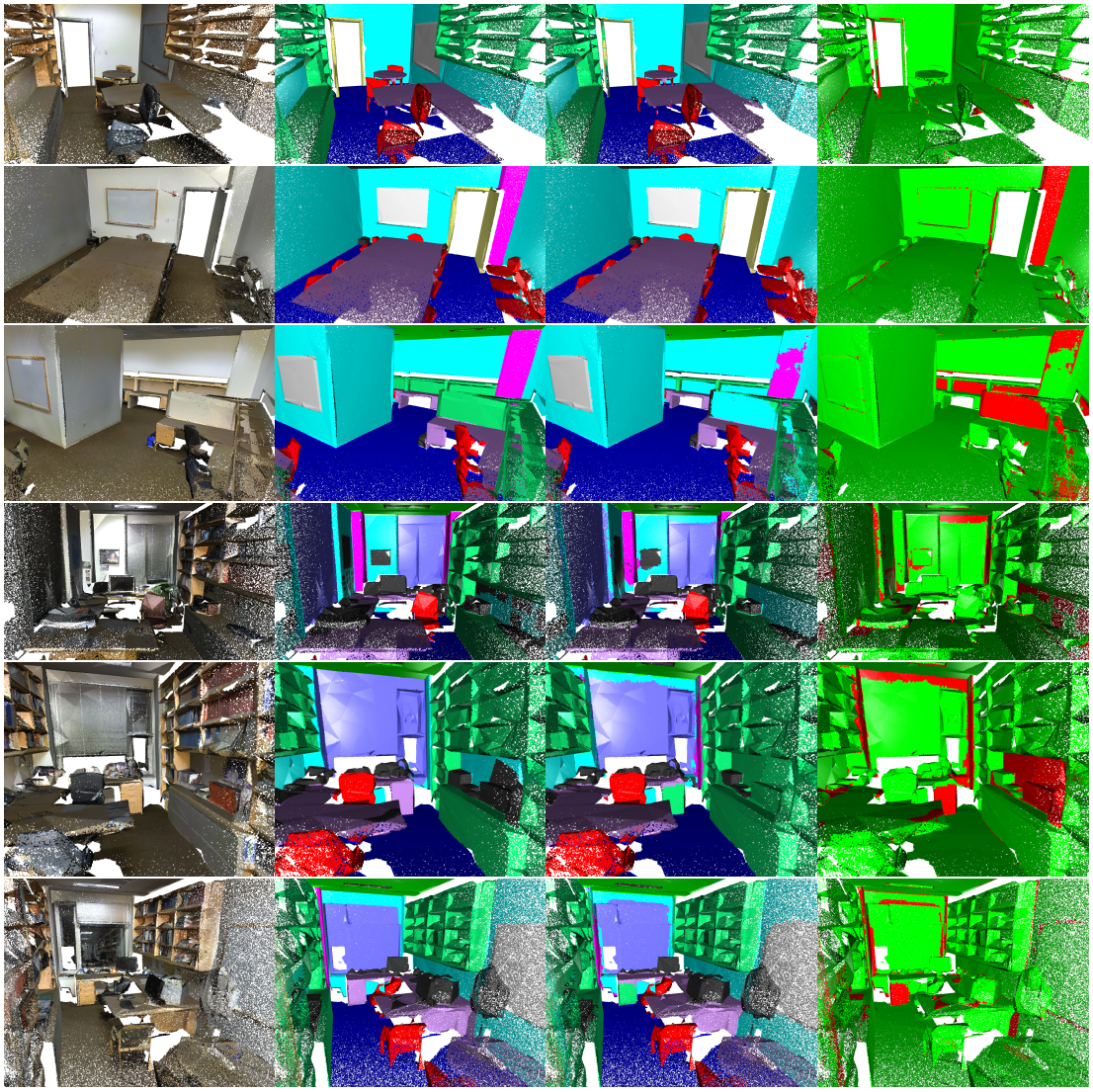


Figure 4.4: **Results on Stanford Large-Scale 3D Indoor Spaces Area 5 [ASZ+16].** Our method correctly predicts challenging classes such as board, while maintaining clear boundaries for most of the classes. In the second row, our method confuses the similar classes column and wall. In Row 4 and Row 5, our method produces unclear boundaries for diverse clutter regions and tends to predict more specific classes such as predicting the books as bookshelf in Row 5 or predicting the backside of the bookshelf as wall in Row 4. Moreover, our algorithm returns reasonable errors. For instance, partition walls between tables are classified as wall and not as bookshelf in Row 3.

5 Ablation Study

We conduct a thorough ablation study in order to support our claims that (1) random edge sampling for effectively sampling the neighborhood space, (2) the combination of geodesic and Euclidean feature extraction, and (3) mesh simplification algorithms as a means of pooling operations, independently contribute to overall improved performance. We conclude with discussing architectural design choices, including the number of graph levels and the choice of the activation functions as well as providing runtime measurements of our best performing model on ScanNet v2.

5.1 Expected Sampling Size

In Chapter 3.3, we have introduced Random Edge Sampling (RES) for randomly sampling subsets of the neighborhood while guaranteeing an upper limit of the expected size of the sampled neighborhood. Therefore, we can vary this limit during train and inference time. In Figure 5.1, we show the relationship between training a network with a relatively small expected sample size and performing inference with other sample sizes. We experience that a small sample size, e.g. $T = 15$, during training is still sufficient to learn useful features of the neighborhood. During inference, we obtain better approximations of the neighborhood with larger thresholds, e.g. $T = 35$, and report significantly better segmentation performances. Therefore, we conclude that RES is a simple yet effective neighborhood sampling algorithm that allows to train more complex models with smaller expected neighborhood sizes while leveraging larger neighborhoods to gain better performances during inference.

All ablation studies which consider DualConvNets using radius neighborhoods leverage random edge sampling. We train the networks with an upper limit of the expected size of the sampled neighborhood of $T = 15$ and evaluate these networks with $T = 25$ as performances gained by RES has shown to saturate after this threshold. Since we deal with a probabilistic sampling method, we evaluate all networks with 10 runs in order to provide the standard deviation for taking fluctuations into account.

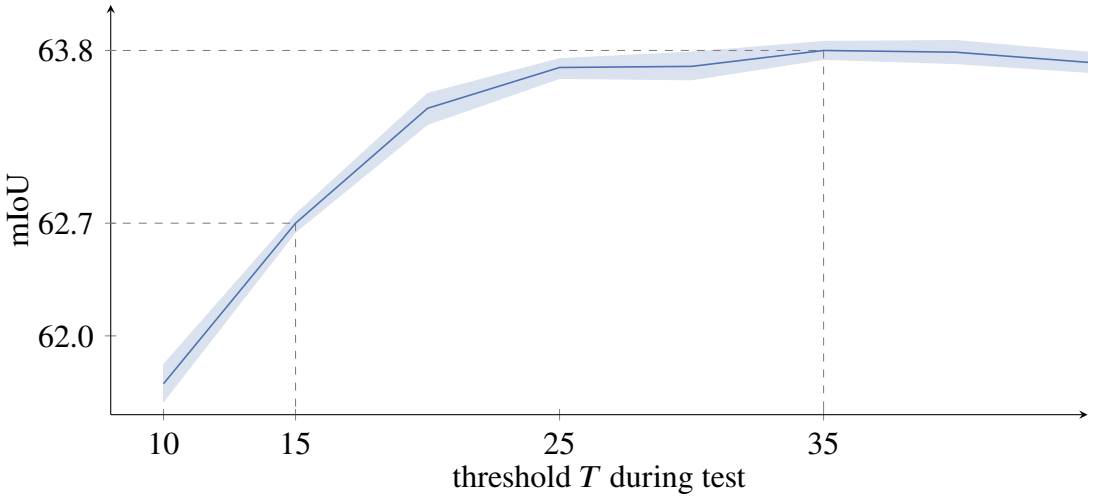


Figure 5.1: **Varying the threshold T during test**. We notice that a smaller number of neighbor samples during training is sufficient to learn useful features ($T = 15$). During inference, we change the upper limit of the sampled neighborhood. We gain 1.1% mIoU by increasing the threshold to $T = 35$. (Experiments conducted on S3DIS Area 5 with 10 runs for each threshold. Light blue area shows standard deviation.)

5.2 Geodesic and Euclidean Convolutions

In this chapter, we discuss the main contribution of this thesis. We show that the combination of geodesic and Euclidean convolutions brings consistent performance gains over operating only in a single space. These improvements are moreover independent of the architecture or pooling operation used.

SingleConvNets vs. DualConvNets. In Chapter 3.1, we have presented dual convolutions which perform graph convolutions in both the Euclidean and geodesic domain in parallel. We refer to an instantiation of our network that only operates in either the geodesic or Euclidean space as a *SingleConvNet*, whereas our full model operating simultaneously in both spaces is referred to as a *DualConvNet*. Technically, SingleConvNets do not use dual convolution modules but directly perform graph convolutions on the designated Euclidean or geodesic neighborhood. Analogously, DualConvNets use dual convolution modules (see Chapter 3.2). Note that SingleConvNets are a subset of the family of DualConvNets since they equal DualConvNet networks if the number of either the Euclidean or geodesic filters is set to 0 everywhere. For each graph convolution in the SingleConvNet architecture, we set the hidden feature size to 128 and the output size to 64. To enable a fair comparison, we halve the hidden and output feature size of DualConvNet architectures, such that the total number of feature channels is equal between the two versions. Note that this results in more than 15% less parameters for DualConvNet architectures (see Table 7.1 and Table 7.2 for precise network definitions in the appendix).

pooling	**architecture**	neighborhood	mIoU	stdev	impr
VC	SingleConvNet	geodesic	57.1		+2.4
VC	SingleConvNet	knn	57.4		+2.1
VC	DualConvNet	knn + geodesic	59.5		
QEM	SingleConvNet	geodesic	56.8		+4.2
QEM	SingleConvNet	knn	57.3		+3.9
QEM	DualConvNet	knn + geodesic	61.2		
VC	SingleConvNet	geodesic	57.1		+5.9
VC	SingleConvNet	radius	62.0	±0.15	+0.7
VC	DualConvNet	radius + geodesic	62.7	±0.16	
QEM	SingleConvNet	geodesic	56.8		+10.4
QEM	SingleConvNet	radius	63.9	±0.14	+3.3
QEM	DualConvNet	radius + geodesic	67.2	±0.17	

Table 5.1: **Effect of combining geodesic and Euclidean Convolutions.** Combining geodesic and Euclidean convolutions in our DualConvNet brings performance improvements, especially compared to solely geodesic convolutions in a SingleConvNet.

architecture	neighborhood	mIoU	stdev	impr
DualConvNet	geodesic + geodesic	56.4		+10.8
DualConvNet	radius + radius	62.7	±0.26	+4.5
DualConvNet	radius + geodesic	67.2	±0.17	
SingleConvNet	radius	63.9	±0.14	−1.2
DualConvNet	radius + radius	62.7	±0.26	
SingleConvNet	geodesic	56.8		−0.4
DualConvNet	geodesic + geodesic	56.4		

Table 5.2: **Influence of the DualConvNet architecture.** We see clear improvements when using geodesic and Euclidean neighborhoods in parallel, in contrast to only using the same neighborhood information for both convolution types of the dual convolution module. QEM is used as the pooling method.

In Table 5.1, we compare models using only geodesic convolutions, only Euclidean convolutions, and both combined in dual convolution modules, while keeping the pooling method fixed. We experience the trend that SingleConvNet architectures using geodesic convolutions fall behind their Euclidean counterparts, whereas the effect for radius neighborhoods is stronger than for k-nn ones.

geodesic/Euclidean ratio				
level 1-2	level 3-4	mIoU	stdev	impr
75%	75%	66.1	±0.11	+2.2
25%	75%	66.1	±0.10	+2.2
25%	25%	66.8	±0.16	+1.5
50%	50%	67.5	±0.15	+0.8
75%	25%	**68.3**	±0.14	

Table 5.3: **Ratio between geodesic and Euclidean filters per graph level.** Geodesic convolutions are particularly useful in early hierarchy levels, when the mesh has not yet undergo much simplification and high frequency signals on the mesh are still preserved. In later hierarchy levels, we apply Euclidean convolutions which take the Euclidean proximity into account and are therefore particularly beneficial for localizing objects within the scene. (Level 1-2 use 64 and level 3-4 use 96 filters in total. See Table 7.4 in the appendix for the full network definition.)

However, the combination of geodesic and k-nn / radius neighborhood in a DualConvNet architecture outperforms both of the SingleConvNet architectures. To additionally prove that these performance gains do not just originate from the change to a DualConvNet architecture, we conducted further experiments in Table 5.2. We see that introducing our DualConvNet architecture leads to worse results in direct comparison with the SingleConvNet architecture when using the same notion of neighborhood for both convolution typess simultaneously. Our assumption is further emphasized by the fact that DualConvNets have a smaller capacity since they comprise 15% parameters than their SingleConvNet counterparts. We thus conclude that the improvements brought by the combination of neighborhoods is based on the design decision of combining geodesic and Euclidean neighborhoods and is not just due to architectural artifacts.

Ratio between Euclidean and geodesic filters. In Chapter 3.2, we have presented our intuition about geodesic and Euclidean convolutions. We assume that geodesic graph convolutions operating directly on surfaces use high-frequency mesh signals for learning features of object-specific shapes. Contrastingly, Euclidean convolutions do not leverage the mesh structure but work on Euclidean neighborhoods defined over the proximity in the Euclidean space. We suppose that this leads to good representations for localizing objects within the scene, e.g. their positioning relative to other objects. Therefore, we claim that geodesic convolutions are particularly useful in shallow network levels since here, the mesh structure is not yet greatly simplified. Euclidean convolutions, however, benefit from simplified mesh structures because the receptive field is enlarged which is particularly helpful for localizing objects.

In order to systematically evaluate our intuition, we conduct experiments where we vary the ratio of geodesic and Euclidean filters per mesh level (see Table 5.3). We see

pooling	architecture	**neighborhood**	mIoU	stdev	impr
VC	SingleConvNet	knn	57.4		+4.6
VC	SingleConvNet	radius	62.0	±0.15	
VC	DualConvNet	knn + geodesic	59.5		+3.2
VC	DualConvNet	radius + geodesic	62.7	±0.16	
QEM	SingleConvNet	knn	57.3		+6.6
QEM	SingleConvNet	radius	63.9	±0.14	
QEM	DualConvNet	knn + geodesic	61.2		+6.0
QEM	DualConvNet	radius + geodesic	67.2	±0.17	

Table 5.4: **Comparison of Euclidean neighborhood notions.** With a significant margin, networks operating on radius neighborhoods outperform their counterparts with k-nn graph neighborhoods in all tested settings. As k-nn approaches are vulnerable to non uniform vertex density distributions [HRV+18], they cannot leverage the potential of QEM pooling approaches and clearly lack behind radius neighborhoods.

that the combination with most geodesic filters in the first two mesh levels and the most Euclidean filters in the last two levels leads to an overall best performance, whereas the exact opposite in ratios leads to worst scores. We see this as a clear indicator that our assumptions about the properties about Euclidean and geodesic convolutions hold.

Neighborhood notion. As we discussed in Chapter 2.2, Euclidean convolutions rely on a notion of locality in order to learn features restricted by Euclidean proximity. Popular neighborhood definitions include k-nn and radius graph neighborhoods. While a fixed neighborhood size characterizes k-nn graphs, radius graphs comprise a constant volume in which neighboring points are considered as neighbors. Hermosilla *et al*. [HRV+18] therefore argue that this increases robustness to a varying density distribution in contrast to k-nn approaches. In Table 5.4, we compare the previously mentioned neighborhood notions for the Euclidean domain. We experience that networks operating with radius neighborhoods outperform their k-nn counterparts with a significant margin in all experiments we have conducted. Especially when performing pooling with Quadric Error Metrics, this result becomes apparent.

5.3 Comparison of Pooling Methods

Our multi-scale hierarchy relies on pooling operators to construct levels of different resolutions. Following a mesh-centric approach, we rely on the previously introduced mesh simplification approaches to implement pooling (see Chapter 3.4). In Table 5.5, we evaluate the influence of different pooling methods in our architecture. We use our

pooling	architecture	neighborhood	mIoU	stdev	impr
VC	SingleConvNet	knn	57.4		−0.1
QEM	SingleConvNet	knn	57.3		
VC	SingleConvNet	geodesic	57.1		−0.3
QEM	SingleConvNet	geodesic	56.8		
VC	SingleConvNet	radius	62.0	±0.15	+1.9
FPS	SingleConvNet	radius	62.6		+1.3
QEM	SingleConvNet	radius	63.9	±0.14	
VC	DualConvNet	knn + geodesic	59.5		+1.7
QEM	DualConvNet	knn + geodesic	61.2		
VC	DualConvNet	radius + geodesic	62.7	±0.16	+4.5
QEM	DualConvNet	radius + geodesic	67.2	±0.17	

Table 5.5: **Comparison of pooling methods.** We compare Vertex Clustering (VC), Farthest Point Sampling (FPS) and Quadric Error Metrics (QEM) as pooling methods for the input mesh/pointcloud. Quadric Error Metrics outperform all other pooling operators even when only considering Euclidean convolutions. We furthermore see that especially the interplay between radius neighborhoods and QEM is beneficial.

extensions of Vertex Clustering (VC) and Quadric Error Metrics (QEM) (see Chapter 3.4) as mesh-centric pooling methods whereas we additionally compare against point cloud based approaches such as Farthest Point Sampling (FPS) [QYSG17] and uniform sampling [TQD+19]. Note that Vertex Clustering equals uniform sampling in the setting of only considering Euclidean neighborhoods since here, we only rely on vertex positions and neglect the mesh interconnectivity. For comparing mesh-based pooling operators against their point cloud counterparts, we only consider SingleConvNets since pooling operators such as FPS do not preserve the underlying mesh structure. Remeshing point clouds after applying FPS is beyond the scope of this thesis.

We conclude that QEM pooling significantly performs better than Vertex Clustering in the DualConvNet architecture, whereas effects are stronger when considering radius neighborhoods for Euclidean convolutions. We trace this back to the fact that QEM does not guarantee a uniform density distribution of vertices while k-nn approaches are vulnerable to varying density distribution as the receptive field might be limited in complex areas of the mesh. Therefore, we argue that particularly the interplay between radius neighborhoods and Quadric Error Metrics is beneficial.

Moreover, we see that QEM outperforms all other pooling methods even when using only Euclidean convolutions with a significant margin of 1.3% to FPS and 1.9% to uniform sampling. We therefore conclude that pooling operators borrowed from the domain of geometry processing have a special appeal. Although not considering the preserved mesh structure, they still outperform point cloud pooling approaches.

#hierarchy levels	mIoU	stdev	impr
2	54.5	±0.13	+12.7
3	63.9	±0.15	+3.3
4	**67.2**	±0.17	

Table 5.6: **Influence of mesh levels.** The number of mesh hierarchy levels has a significant impact of the performance of DualConvNets. With a decreasing effect, more mesh levels consistently leads to better results. (We use QEM pooling and radius + geodesic neighborhoods in the DualConvNet architecture for this experiments.)

activation function	mIoU	stdev	impr
Leaky ReLU	65.8	±0.11	+1.4
ReLU	**67.2**	±0.17	

Table 5.7: **Comparison of activation functions.** As Leaky ReLU gains popularity in the field, we compare it against standard ReLU activation functions. We see that default ReLU units outperform LeakyReLU activation functions by a margin of 1.4%. (Experiments were conducted with QEM pooling and radius + geodesic neighborhoods in our DualConvNet architecture.)

5.4 Architectural Design Choices

In the previous chapter, we have evaluated architecture agnostic components of our method. In this chapter, however, we give more details about our architectural design choices: We show the impact of the number of graph levels for the DualConvNet architecture as well as comparing different activation functions in our architecture.

Number of mesh levels. Multi-scale hierarchies have gained popularity in the field since they make feature extraction at various resolutions possible [TQD+19,GEvdM18, HRV+18]. At high resolutions, the receptive field of each convolution is restricted but high-frequency signals are used to obtain feature representations. Contrastingly, at lower resolutions, learned features are no longer highly localized but contain more contextual information since their receptive fields are enlarged. In Chapter 3.4, we have extended mesh simplification algorithms in order to build multi-scale architectures on meshes. In Table 5.6, we experimentally evaluate the impact of the number of mesh levels. We see a clear trend that an increased number of hierarchy levels leads to better segmentation results.

Activation functions. Recent publications for 3D semantic scene segmentation such as the work from Thomas *et al*. [TQD+19] rely on LeakyReLU [XWCL15] activation functions for avoiding standard ReLU's *stuck at zero problems*. In Table 5.7,

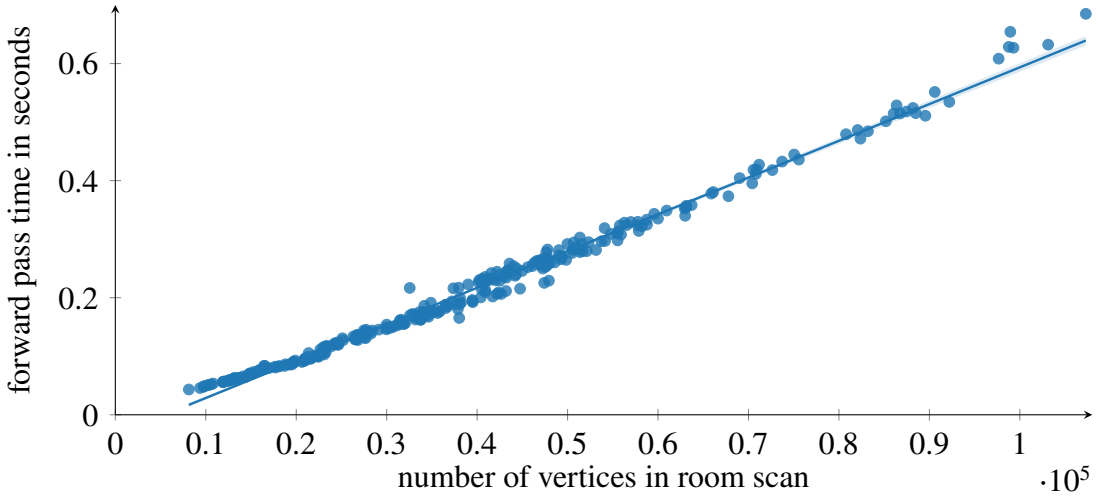


Figure 5.2: **Runtime with respect to the number of vertices**. We observe a piece-wise linear relationship between runtime of forward passes for full rooms of the ScanNet v2 validation set and the number of vertices in the input mesh. (We use our best performing model on ScanNet given in Table 7.4.)

we compare standard ReLU with LeakyReLU activation functions. We conclude that DualConvNets work better with standard ReLU activation functions by a margin of 1.4% mIoU.

5.5 Runtime

Since our method is translation-equivariant (see Chapter 4.3), we can evaluate on full rooms which helps to understand the global context of a scene better. Since we do not rely on merging crops as a postprocessing step, full room evaluation comes with an easier pipeline and an improved runtime.

In Figure 5.2, we provide forward pass times for our best performing ScanNet benchmark model concerning the number of vertices in the input mesh. We see a linear relationship between the number of input vertices and the runtime which is always well under 0.7 seconds for all scans (excluding the preprocessing time). Overall, the mean runtime for the ScanNet validation set is 211ms with an average input size of 39161 vertices. We perform this experiment on a Tesla V100 16GB.

6 Discussion

In this thesis, we have motivated a mesh-centric view on 3D semantic segmentation of indoor scenes where we learn feature representations based on both the mesh surface as well as the relative positioning of vertices in the Euclidean domain. In order to prove that combining geodesic and Euclidean features leads to better segmentation performances, we have proposed a novel architecture family which we called *DualConvNets*. Here, we have proposed *dual convolution modules* which simultaneously perform convolutions in both the geodesic space along the mesh in order to learn features focusing on the surface structure as well as in the Euclidean space which enables us to learn the interaction between disconnected objects.

In order to leverage the potential of multi-scale architecture, we need to preserve surface structures of subsequent mesh levels. We extend mesh simplification algorithms such as Vertex Clustering and Quadric Error Metrics to gradually simplify the mesh while maintaining approximate surface representations for geodesic convolutions. We efficiently interlink mesh levels by using *pooling trace maps* which represent simple look-up dictionaries to obtain representative vertices in consecutive mesh levels.

As a technical contribution, we have proposed *Random Edge Sampling* which randomly samples Euclidean neighborhood sets while guaranteeing an upper limit for the expected size of the sampled set. We thus can conduct experiments with varying thresholds in train and inference time and can reduce the computational load of the algorithm.

We hope that our work encourages fellow researchers to consider convolutions in both the geodesic and Euclidean domain as we can show that this results in a consistent performance gain independent of the architecture used.

6.1 Open Challenges and Future Work

In this chapter, we give some impulses for further thoughts about open challenges of DualConvNets. Additionally, we point out what future research directions might include which extend our work.

Overlap of geodesic and Euclidean edge set. For each vertex in the mesh, we consider two neighborhood sets, namely the Euclidean set which consists of vertices in the Euclidean proximity of the vertex and the geodesic set which comprises 1-hop neighbors defined by the inherent surface mesh structure. However, these two sets are correlated. For a given vertex, vertices in the geodesic neighborhood set tend to be in the Euclidean proximity, as well. Therefore, it is likely that these vertices appear in both the geodesic and Euclidean neighborhood set. This overlap poses a computational burden in terms of memory consumption since there exists a significant overlap between both sets. Computations are then not only performed on vertices which are characteristic for defining the specific Euclidean or geodesic set. Thus, future work may focus on the question of how to better separate both sets, finding the characteristic, disjunctive subset for both the Euclidean and geodesic set in order to slim down the training process without losing information.

Extending submanifold sparse convolutions. Variants of submanifold sparse convolutions such as SparseConvNet [GEvdM18] and MinkowskiNet [CGS19] have recently defined new state-of-the-art performance on various semantic segmentation tasks for 3D indoor scenes. However, instead of leveraging surface information defined by the mesh structure, these approaches operate on point clouds only thus leaving room for improvement. In our work, we showed that the combination of geodesic and Euclidean convolutions in dual convolution modules brings performance gains. In the following, we thus present two orthogonal ideas for future work of how to combine sparse convolutions with geodesic convolutions in a mesh-centric setting. At the center of our attention, we focus on the mapping of continuous vertices for geodesic convolutions with their corresponding voxels for discrete Euclidean convolutions in a dual convolution module in order to transfer features between both convolution types.

(1) **Pooling-based one-to-one mapping**. Sparse convolutional approaches discretize the Euclidean domain in a grid-like fashion. However, original vertices are embedded in the continuous space. In order to apply dual convolutions, we establish a bijective (1-to-1) mapping between voxels in the discretized domain with their corresponding continuous vertices for concatenating unique vertex representations between both domains. Using vertex clustering, we can guarantee bijectivity between continuous vertices and their corresponding voxels since each voxel is associated with its continuous center of gravity based on the vertices falling into it. Contrarily, Quadric Error Metrics does not guarantee the bijectivity of the voxel-vertex mapping. For instance, during the simplification process, new continuous representatives are generated which might be placed in already occupied voxels and thus breaking bijectivity. Since vertex clustering has shown to be inferior to QEM pooling, we need to adapt the latter one to preserve bijectivity. We first apply vertex clustering on the input mesh. Thus, generating an initial bijective voxel-vertex mapping. Instead of finding the continuous representative r which minimizes the contraction cost $\Delta(r)$ (Equation 2.9), we consider only the endpoints of the potential contraction and choose the one with minimal cost. Here, we know to which voxel it belongs to and we hence maintain the bijectivity of the voxel-

vertex mapping. We call this approach pooling-based since the mesh simplification algorithm generates new continuous representatives while discrete voxels are subsequently maintained accordingly.

② **Aggregation-based many-to-one mapping.** Alternatively, we can perform pooling independently in the discrete and continuous space. Here, inconsistencies might occur, such as many-to-one mappings when multiple vertices fall into the same voxel. We resolve these issues by aggregating the variable number of vertices in the same voxel in a permutation-invariant manner to receive a defined feature size for concatenating them with Euclidean features from the sparse convolutions.

Geodesic convolutions for refining instance segmentations. Geodesic convolutions enlarge their receptive field only along the object's surface which leads to highly object-dependent features. Fusing features of nearby but geodesically disconnected objects is thus not possible. This characteristic behavior of geodesic convolutions justify their special appeal for instance segmentation since separating objects from each other is drastically simplified. Therefore, we suspect that investigating geodesic convolutions for 3D instance segmentation is a promising research directions.

7 Appendix

The appendix provides detailed architecture descriptions for the ablation study (Table 7.1 and Table 7.2) as well as the adapted networks for the official benchmarks on Stanford Large-Scale 3D Indoor Spaces (Table 7.3), Matterport3D (Table 7.4) and ScanNet v2 Benchmark (Table 7.4).

#level	level type	module type	filters
1	encoder	edge+BN+ReLU	(9, 128, 64)
1	encoder	edge+BN+ReLU	(128, 128, 64)
1	encoder	edge+BN+ReLU	(128, 128, 64)
2	encoder	edge+BN+ReLU	(128, 128, 64)
2	encoder	edge+BN+ReLU	(128, 128, 64)
2	encoder	edge+BN+ReLU	(128, 128, 64)
3	encoder	edge+BN+ReLU	(128, 128, 64)
3	encoder	edge+BN+ReLU	(128, 128, 64)
3	encoder	edge+BN+ReLU	(128, 128, 64)
4	encoder	edge+BN+ReLU	(128, 128, 64)
4	encoder	edge+BN+ReLU	(128, 128, 64)
4	encoder	edge+BN+ReLU	(128, 128, 64)
3	decoder	edge+BN+ReLU	(256, 128, 64)
3	decoder	edge+BN+ReLU	(128, 128, 64)
3	decoder	edge+BN+ReLU	(128, 128, 64)
2	decoder	edge+BN+ReLU	(256, 128, 64)
2	decoder	edge+BN+ReLU	(128, 128, 64)
2	decoder	edge+BN+ReLU	(128, 128, 64)
1	decoder	edge+BN+ReLU	(256, 128, 64)
1	decoder	edge+BN+ReLU	(128, 128, 64)
1	decoder	edge+BN+ReLU	(128, 128, 64)
1	final	Lin+BN+ReLU	(64, 32)
1	final	Lin	(32, 21)
		# parameters:	**564, 949**

Table 7.1: **SingleConvNet architecture for the ablation study.** For the ablation study, we use the SingleConvNet architecture. Here, we do not use dual convolution modules but directly apply edge convolutions (edge) with 128 hidden and 64 output features on the designated Euclidean or geodesic neighborhood. To reduce the feature size to the number of classes $C = 21$ we apply 2 linear transformations. We use batch normalization (BN) and ReLU activation functions.

#level	level type	module type	filters
1	encoder	edge+BN+ReLU	$2 * (9, 64, 32)$
1	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
1	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
2	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
2	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
2	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
3	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
3	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
3	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
4	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
4	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
4	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
3	decoder	edge+BN+ReLU	$2 * (256, 64, 32)$
3	decoder	edge+BN+ReLU	$2 * (128, 64, 32)$
3	decoder	edge+BN+ReLU	$2 * (128, 64, 32)$
2	decoder	edge+BN+ReLU	$2 * (256, 64, 32)$
2	decoder	edge+BN+ReLU	$2 * (128, 64, 32)$
2	decoder	edge+BN+ReLU	$2 * (128, 64, 32)$
1	decoder	edge+BN+ReLU	$2 * (256, 64, 32)$
1	decoder	edge+BN+ReLU	$2 * (128, 64, 32)$
1	decoder	edge+BN+ReLU	$2 * (128, 64, 32)$
1	final	Lin+BN+ReLU	$(64, 32)$
1	final	Lin	$(32, 21)$
		# parameters:	**478, 933**

Table 7.2: **DualConvNet architecture for the ablation study.** In the DualConvNet architecture, we leverage dual convolution modules which perform convolutions simultaneously in both the geodesic and Euclidean space and subsequently concatenate the features. Note that the total size of hidden and output features for each dual convolution equals its SingleConvNet's edge convolution equivalent (see Table 7.1). We are therefore able to perform fair comparisons between SingleConvNets and DualConvNets. To reduce the feature size to the number of classes $C = 21$ we apply 2 linear transformations. We use batch normalization (BN) and ReLU activation functions.

#level	level type	module type	filters
1	encoder	edge+BN+ReLU	$2 * (9, 64, 32)$
1	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
1	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
2	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
2	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
2	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
3	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
3	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
3	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
4	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
4	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
4	encoder	edge+BN+ReLU	$2 * (128, 64, 32)$
3	decoder	edge+BN+ReLU	$2 * (384, 96, 48)$
3	decoder	edge+BN+ReLU	$2 * (192, 96, 48)$
3	decoder	edge+BN+ReLU	$2 * (192, 96, 48)$
2	decoder	edge+BN+ReLU	$2 * (320, 64, 32)$
2	decoder	edge+BN+ReLU	$2 * (128, 64, 32)$
2	decoder	edge+BN+ReLU	$2 * (128, 64, 32)$
1	decoder	edge+BN+ReLU	$2 * (256, 64, 32)$
1	decoder	edge+BN+ReLU	$2 * (128, 64, 32)$
1	decoder	edge+BN+ReLU	$2 * (128, 64, 32)$
1	final	Lin+BN+ReLU	$(64, 32)$
1	final	Lin	$(32, 13)$
		# parameters:	**728, 045**

Table 7.3: **Architecture used for obtaining the final scores on the S3DIS benchmark.** In contrast to the DualConvNet used for the ablation study, we use more filters in the final two graph levels. As we have discussed in Chapter 4.1, the mesh resolution of S3DIS is low. We therefore do not benefit from increasing the number of geodesic filters in early levels and the number of Euclidean filters in later levels. So, we set the ratio to 50% for both types in each level.

			filters	
#level	level type	module type	geodesic	Euclidean
1	encoder	edge+BN+ReLU	(9, 96, 48)	(9, 32, 16)
1	encoder	edge+BN+ReLU	(128, 96, 48)	(128, 32, 16)
1	encoder	edge+BN+ReLU	(128, 96, 48)	(128, 32, 16)
2	encoder	edge+BN+ReLU	(128, 96, 48)	(128, 32, 16)
2	encoder	edge+BN+ReLU	(128, 96, 48)	(128, 32, 16)
2	encoder	edge+BN+ReLU	(128, 96, 48)	(128, 32, 16)
3	encoder	edge+BN+ReLU	(128, 48, 24)	(128, 144, 72)
3	encoder	edge+BN+ReLU	(192, 48, 24)	(192, 144, 72)
3	encoder	edge+BN+ReLU	(192, 48, 24)	(192, 144, 72)
4	encoder	edge+BN+ReLU	(192, 48, 24)	(192, 144, 72)
4	encoder	edge+BN+ReLU	(192, 48, 24)	(192, 144, 72)
4	encoder	edge+BN+ReLU	(192, 48, 24)	(192, 144, 72)
3	decoder	edge+BN+ReLU	(384, 48, 24)	(384, 144, 72)
3	decoder	edge+BN+ReLU	(192, 48, 24)	(192, 144, 72)
3	decoder	edge+BN+ReLU	(192, 48, 24)	(192, 144, 72)
2	decoder	edge+BN+ReLU	(320, 96, 48)	(320, 32, 16)
2	decoder	edge+BN+ReLU	(128, 96, 48)	(128, 32, 16)
2	decoder	edge+BN+ReLU	(128, 96, 48)	(128, 32, 16)
1	decoder	edge+BN+ReLU	(256, 96, 48)	(256, 32, 16)
1	decoder	edge+BN+ReLU	(128, 96, 48)	(128, 32, 16)
1	final	Lin+BN+ReLU	(64, 32)	
1	final	Lin	(32, C)	

ScanNet	# parameters:	**761, 333**
Matterport3D	# parameters:	**761, 366**

Table 7.4: **Architecture used for obtaining the final scores on ScanNet and Matterport3D.** In contrast to the DualConvNet used for the ablation study, we use more filters in later two graph levels and use the best performing filter ratio from Table 5.3. We obtain different numbers of parameters for ScanNet and Matterport3D since they differ in their number of labeled classes ($C_{\text{scannet}} = 21$ and $C_{\text{matterport}} = 22$).

Bibliography

[AML18] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point Convolutional Neural Networks by Extension Operators. *ACM Transactions on Graphics (TOG)*, 2018.

[ASZ+16] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3D Semantic Parsing of Large-Scale Indoor Spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[BBL+17] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 2017.

[BHB+18] Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018.

[CBL+19] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[CDF+17] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017.

[CGS19] Christopher Choy, Jun Young Gwak, and Silvio Savarese. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[CLLH19] Hungyueh Chiang, Yenliang Lin, Yuehcheng Liu, and Winston H Hsu. A unified point-based framework for 3d segmentation. *International Conference on 3D Vision (3DV)*, 2019.

[DCS+17] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[DJJ+18] Rethage Dario, Wald Johanna, Sturm Juergen, Navab Nassir, and Tombari Federico. Fully-Convolutional Point Networks for Large-Scale Point Clouds. In *IEEE European Conference on Computer Vision (ECCV)*, 2018.

[DN18] Angela Dai and Matthias Nießner. 3DMV: Joint 3D-Multi-View Prediction for 3D Semantic Scene Segmentation. In *IEEE European Conference on Computer Vision (ECCV)*, 2018.

[EKL20] Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. Dilated point convolutions: On the receptive field of point convolutions. *IEEE International Conference in Robotics and Automation (ICRA)*, 2020.

[FL19] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[GEvdM18] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using Quadric Error Metrics. In *Computer Graphics and Interactive Techniques*, 1997.

[GSR+17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *International Conference on Machine Learning (ICML)*, 2017.

[GWL18] Fabian Groh, Patrick Wieschollek, and Hendrik P. A. Lensch. Flex-convolution (million-scale point-cloud learning beyond grid-worlds). In *Asian Conference on Computer Vision (ACCV)*, 2018.

[HAB+17] Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the kepler conjecture. In *Forum of Mathematics, Pi*, 2017.

[HRV+18] Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Alvar Vinacua, and Timo Ropinski. Monte Carlo Convolution for Learning on Non-Uniformly Sampled Point Clouds. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2018)*, 2018.

[HS17] Jaber J. Hasbestan and Inanc Senocak. Binarized octree generation for cartesian adaptive mesh refinement around immersed geometries. *Journal of Computational Physics*, 2017.

[HSYX18] Pan Hao, Liu Shilin, Liu Yang, and Tong Xin. Convolutional Neural Networks on 3D Surfaces Using Parallel Frames. *arXiv preprint arXiv:1808.04952*, 2018.

[HTY18] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Point-wise Convolutional Neural Network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[HWN18] Qiangui Huang, Weiyue Wang, and Ulrich Neumann. Recurrent slice networks for 3d segmentation of point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[HZY+19] Jingwei Huang, Haotian Zhang, Li Yi, Thomas A. Funkhouser, Matthias Nießner, and Leonidas J. Guibas. TextureNet: Consistent Local Parametrizations for Learning from High-Resolution Signals on Meshes. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[IS15] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning (ICML)*, 2015.

[JGS19] Maximilian Jaritz, Jiayuan Gu, and Hao Su. Multi-view pointnet for 3d scene understanding. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, 2019.

[JZL+19] Li Jiang, Hengshuang Zhao, Shu Liu, Xiaoyong Shen, Chi-Wing Fu, and Jiaya Jia. Hierarchical point-edge interaction network for point cloud semantic segmentation. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations, (ICLR)*, 2015.

[KM17] Eamonn Keogh and Abdullah Mueen. *Curse of Dimensionality*, pages 314–315. Springer US, Boston, MA, 2017.

[KUH+19] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Lyft level 5 av dataset 2019. urlhttps://level5.lyft.com/dataset/, 2019.

[KW17] Thomas Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations, (ICLR)*, 2017.

[LAM19] Huan Lei, Naveed Akhtar, and Ajmal Mian. Spherical Kernel for Efficient Graph Convolution on 3D Point Clouds. *arXiv preprint arXiv:1909.09287*, 2019.

[LBS+18] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution On X-Transformed Points. In *Neural Information Processing Systems (NIPS)*, 2018.

[LMTG19] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[LS18] Landrieu Loic and Martin Simonovsky. Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[MBM+17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[MS15] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D Convolutional Neural Network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[PGC+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Neural Information Processing Systems Workshop (NIPSW)*, 2017.

[QLJ+17] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. 3D Graph Neural Networks for RGBD Semantic Segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[QSMG17] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[QYSG17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Neural Information Processing Systems (NIPS)*, 2017.

[RB93] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in Computer Graphics*, 1993.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[SC14] Tibor Stanko and Pavel Chalmovianský. Refining procedures on mesh via algebraic fitting. In *Proceedings of CESCG*, 2014.

[SHK+14] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. In *Journal of Machine Learning Research (JMLR)*, 2014.

[SJS+18] 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 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[SK17] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[SKD+19] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi,

Yu Zhang, Jon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: An open dataset benchmark. *arXiv preprint arXiv:1912.04838*, 2019.

[TCA+17] Lyne P. Tchapmi, Christopher B. Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *International Conference on 3D Vision (3DV)*, 2017.

[TPKZ18] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent Convolutions for Dense Prediction in 3D. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[TQD+19] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[VBK16] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *International Conference on Learning Representations (ICLR)*, 2016.

[VBV18] Nitika Verma, Edmond Boyer, and Jakob Verbeek. FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[WH18] Yuxin Wu and Kaiming He. Group normalization. In *IEEE European Conference on Computer Vision (ECCV)*, 2018.

[WQL19] Wenxuan Wu, Zhongang Qi, and Fuxin Li. PointConv: Deep Convolutional Networks on 3D Point Clouds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[WSL+19] 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)*, 2019.

[WSM+18] S. Wang, S. Suo, W.C. Ma, A. Pokrovsky, and R. Urtasun. Deep Parametric Continuous Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[WYHN18] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. SGPN: Similarity Group Proposal Network for 3D Point Cloud Instance Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[XFX+18] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters. In *IEEE European Conference on Computer Vision (ECCV)*, 2018.

[XWCL15] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network. *arXiv preprint arXiv:1505.00853*, 2015.

[YLH+18] Xiaoqing Ye, Jiamao Li, Hexiao Huang, Liang Du, and Xiaolin Zhang. 3D Recurrent Neural Networks with Context Fusion for Point Cloud Semantic Segmentation. In *IEEE European Conference on Computer Vision (ECCV)*, 2018.

[ZGWmC19] Kai Zhao, Shanghua Gao, Wenguan Wang, and Ming ming Cheng. Optimizing the F-measure for threshold-free salient object detection. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[ZLU18] Chris Zhang, Wenjie Luo, and Raquel Urtasun. Efficient convolutions for real-time semantic segmentation of 3d point clouds. In *International Conference on 3D Vision (3DV)*, 2018.