Doctoral Dissertation

Doctoral Program in Electronics and Telecommunication Engineering (35thcycle)

Learning robust and efficient point cloud representations

By

Francesca Pistilli

******

Supervisor(s):

Prof. Enrico Magli, Supervisor

Prof. Diego Valsesia, Co-Supervisor

Politecnico di Torino

2023

Declaration

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

I would like to dedicate this thesis to my family and friends, for all the support through these years.

Acknowledgements

I would like to express my deep gratitude to my advisor, Professor Enrico Magli, for providing me with the invaluable opportunity of pursuing a Ph.D. degree. I would like to thank him for his assistance, insightful advices, and mentorship throughout my academic journey. I would also like to greatly thank Giulia Fracastoro and Diego Valsesia for their constant unwavering support and guidance, they played a important and inspiring role during these years and helped me to grow as a researcher.

A heartfelt appreciation goes to all my friends and colleagues from the IPL group for their help and support during these years.

Abstract

Point cloud processing is a crucial task in the field of computer vision and signal processing. With the recent increase in 3-D sensing technology, large-scale 3-D point clouds have become more and more available, providing a rich and comprehensive representation of the geometry of real-world objects and scenes. However, processing point clouds poses several challenges due to the irregular and unstructured nature of the data, the imperfect acquisition methods and heavy memory requirements.

Feature extraction is a key challenge for point cloud processing, involving the identification and description of important geometric and semantic features of a point cloud. Dealing with the irregular and unstructured nature of the data makes traditional processing techniques less effective or not applicable. Therefore, new techniques such as geometric deep learning methods and graph-based representations have been developed. Moreover, it has to be considered that all available techniques able to acquire point clouds insert non-negligible noise into the data, changing the global shape of the objects or scene. Therefore, it is particularly important design deep learning methods able to restore the original data and to extract robust features from noisy data. Notably, point clouds are characterized by a large amount of 3-D points with additional attribute information, such as colors or normals to the surface. The huge memory requirements are a critic aspect when it comes to processing this type of data, and there has been a growing interest and necessity in founding alternative memory-efficient data representation.

This thesis is focused on tackling and analyzing the critic aspects previously introduced and two main themes are investigated: i) learning robust graph convolutional features for point cloud processing in the presence of noise and ii) learning an efficient and compact representation for point cloud attributes.

Regarding the first topic, a novel graph convolutional neural network is investigated to learn meaningful features from raw noisy data. The focus of the research is to prove the suitability of the graph convolutional neural network for point cloud processing tasks, particularly in the presence of noisy data. Moreover, the results demonstrate the power of the proposed network for a restoration task such as denoising.

Concerning the second topic, an efficient point cloud compression algorithm is investigated to tackle the problem of memory occupation. Compression algorithms can reduce the size of point cloud data, improving processing time and overall performance while reducing costs associated with handling large data sets. In this thesis, in particular the task of point cloud attribute compression is analyzed, and a novel signal compression algorithm based on neural implicit representations is proposed.

Overall, this thesis proposes novel deep learning-based techniques to address some of the main challenges in point cloud processing and to improve tasks such as denoising, surface normal estimation, and compression. The addressed problems are usually part of a processing pipeline. For instance, denoising is typically a pre-processing procedure used to improve downstream tasks, such classification, segmentation or compression. The research demonstrates the power of graph convolutional neural networks for point cloud processing tasks and provides a new approach for point cloud compression.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Learning representations for point cloud processing

A point cloud is an unordered set of points in 3-D coordinate system that represents the geometry of an object or an entire scene. Each point of the cloud is defined by its 3-D Cartesian coordinates (x, y, z) and by additional attributes such as color, reflectance, surface normals or intensity, see few visual example in Fig. 1.1.

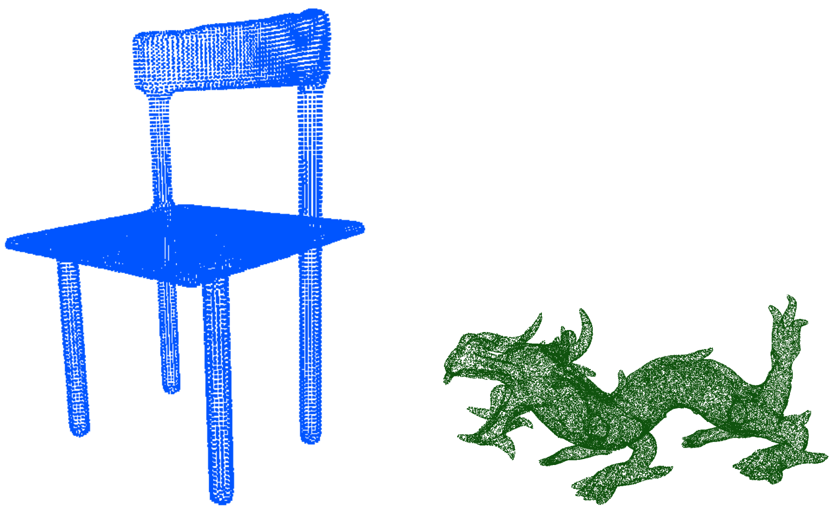


Fig. 1.1 Few visual example of well-known point clouds

Point clouds are commonly used in computer vision, robotics, and 3-D sensing, therefore point cloud processing is a crucial and intriguing task in the field of computer vision and signal processing. In recent years, there has been a rapid increase of 3-D sensing technology, leading to a tremendous increment in the availability of large-scale 3-D point clouds. This type of data provides a rich and comprehensive representation of the geometry of real-world objects and scenes, becoming an essential tool in many applications, including virtual and augmented reality, robotics, and autonomous systems.

However, processing point clouds is not straightforward and it poses several challenges due to the irregular nature of the data, the imperfect acquisition methods and the large amount of data.

One of the key challenges in point cloud processing is to extract meaningful and useful information from the raw data. This task, called feature extraction or representation learning, involves identifying and describing important geometric and semantic features of the point cloud, which can be used for tasks such as object recognition, registration, and tracking. Another challenge in point cloud processing is to deal with the irregular nature of the data, which makes traditional processing techniques, designed for structured data such as images and videos, less effective or not applicable. This has motivated the development of new techniques, such as deep learning methods. There are different types of networks able to process point clouds and they can be roughly divided into projection-based[4, 5], voxel-based [6, 7] and point-based methods [8, 9]. Projection-based methods project points clouds into 2-D images in order to use 2-D convolutional neural networks to conduct various analyses. Such approaches introduces some distortion and artifacts in the shape of the point clouds, leading to sub-optimal results in several tasks. Voxel-based methods perform a grid partition of each point cloud in order to obtain a regular 3-D structure compatible with classic 3-D convolutional networks. Despite the inferred regular structure, the performance of such methods highly rely on the resolution of the voxels. Furthermore, both voxel and projection based methods need a post-processing procedure for regression tasks, since such methods do not process all the points of the original point cloud. Instead, point-based methods directly encode the 3-D positions of raw point clouds. Recently, among this last category, graph-based representations have emerged as a promising approach which can handle the complex and dynamic geometry of point clouds.This type of network is able to naturally deal with the irregular structure of the data and able to uncover latent local and non-local similarities.

Moreover, it has to be considered that all acquisition methods are imperfect and insert a non-negligible amount of noise into the data, changing the global shape of the objects or scenes. Therefore, it is of paramount importance first design deep learning methods able to restore the original data and then methods able to extract robust features from noisy data.

Another critical aspect of point clouds processing is the large amount of data, characterized by 3-D positions with additional attribute information, such as colors or normals to the surface. Therefore, point clouds can be substantial in size and complexity, making storage, transmission, processing and rendering challenging. However, compression algorithms can mitigate these challenges by reducing the size of the point cloud data, improving processing time and overall performance, and reducing costs associated with handling large data sets.

This thesis is focused on two main themes: learning robust graph convolutional features for point cloud processing in presence of noise and learning an efficient and compact representation for point cloud attributes.

In the first topic of this thesis, a novel powerful graph convolutional neural network is investigated to learn meaningful features from raw data. The primary focus of the research is to prove the suitability of such network for a restoration task as denoising. Furthermore, the ability of the proposed method is tested in different challenging scenarios, such in presence of outliers, and for different tasks.Thanks to the remarkable results obtained in our research, it is possible to demonstrate that graph convolutional neural networks are the most promising and powerful tool for point cloud processing tasks, especially in presence of noisy data.

Concerning the second topic, an efficient point cloud compression algorithm is investigated to tackle the problem of memory occupation. In particular, in this thesis the task of point cloud attribute compression in analyzed and a novel signal compression algorithm based on neural implicit representations is proposed. In the proposed paradigm the network, that is trained to fit the desired signal, can be interpreted as the signal itself and it is able to provide a novel and compact representation.

1.2 Thesis organization

The thesis is organized as follows.

In Chapter 2 a general background on deep neural networks is provided, with a special interest regarding graph-convolutional neural networks and implicit neural representations. Moreover, the point cloud processing tasks addressed throughout the thesis are introduced, focusing on the description of the problems and the main motivations.

Chapter 3 starts to investigate deep learning methods to extract meaningful features from point cloud data and proposes a novel graph-convolutional architecture to tackle the task of point cloud denoising called GPDNet [10].

In Chapter 4, an extension of GPDNet, called GPOD [11], is presented. In this research more complex scenarios are investigated, and a unique network able to simultaneously tackle the task of outlier removal and denoising without any prior knowledge about the inserted noise is proposed.

In Chapter 5 the robustness and representation ability of graph convolutional neural networks are further explored to extract geometric features in presence of noise. In particular, the task of surface normal estimation is analyzed and a novel network based on graph convolutional layers is proposed [12].

Following, another essential aspect for point cloud processing is analyzed: founding an efficient learnable point cloud representation. In Chapter 6, the signal compression task is considered and a novel algorithm based on implicit neural representation [13] able to compress any type of signal is presented. The methodology is then applied to the task of point cloud attribute compression.

Finally, in Chapter 7 final conclusions are reported and open questions are discussed.

1.3 List of publications

In this section a list of publications is presented, which represents that well represent the outcome of the research work carried out during the PhD program:

1. F. Pistilli, G. Fracastoro, D. Valsesia, E. Magli. “Learning Graph-Convolutional Representationsfor Point Cloud Denoising”. In *The European Conference on Computer Vision (ECCV)*, 2020.

2. F. Pistilli, G. Fracastoro, D. Valsesia, E. Magli. “ Point Cloud Normal Estimation with Graph-Convolutional Neural Networks”. In *2020 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*,pages 1-6, 2020.

3. F. Pistilli, G. Fracastoro, D. Valsesia, E. Magli. “ Learning Robust Graph-Convolutional Representationsfor Point Cloud Denoising”. *IEEE Journal of Selected Topics in Signal Processing*, volume 15, pages 402-414, 2021.

4. F. Pistilli, D. Valsesia, G. Fracastoro, E. Magli. “ Signal Compression via Neural Implicit Representations”. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3733-3737, 2022.

Chapter 2

Background

2.1 Deep Learning for Point Cloud Processing

Deep learning is a subset of machine learning that is concerned with the development of artificial neural networks that can learn from data, and perform a wide range of tasks such as image recognition, speech recognition, and natural language processing. One of the main characteristics of such networks is attributed to the fact that they are composed by multiple layers of artificial neurons, which enables them to learn increasingly complex features of the data, as the information is processed through each layer.

Recently, deep learning has played a significant impact on point cloud processing. Before, point cloud processing was typically performed using traditional methods such as geometric algorithms [14–16], feature-based methods [17–19], and template matching [20, 21].

The first big impact of deep learning to point cloud processing is the development of PointNet [8], which is a neural network architecture specifically designed for point cloud data. PointNet uses a shared-weight architecture to process all points independently, allowing it to handle unordered point sets. This architecture has been used for various tasks such as classification, semantic segmentation and 3-D object detection. Later, few extensions of PointNet has been proposed, such PointNet++[9] that introduces a hierarchical structure, recursively applying a PointNet-like architecture on patches extracted from the input point cloud.

Recently, Graph Neural Networks (GNNs) have emerged as a promising technique to process point clouds in an unordered way by using graph structures that represent the relationships between points. GNNs have been applied to tasks such as object classification and semantic segmentation.

Overall, deep learning has greatly improved the performance of point cloud processing, and have achieved many state-of-the-art results for several challenging applications.

2.1.1 Neural Networks

A neural network is a type of machine learning model that is inspired by the structure and function of the human brain. The fundamental building block of neural networks is the “neuron” that receives inputs, performs a simple computation on them, and produces an output. The inputs of a neuron are multiplied by a set of weights, the sum of these weighted inputs is then passed through an activation function, which is used to introduce non-linearity into the network generating the output, see Fig. 2.1.

A fully-connected neural network, the simplest neural network architecture, is composed by several layers, each constituted by multiple neurons. Each layer takes as input the output of the previous layer and provides as output a new feature representation of the input:

$$\mathbf{h}^{l+1} = \sigma(\mathbf{W}^{l+1,T}\mathbf{h}^{l} + \mathbf{b}^{l+1}), \tag{2.1}$$

where $\mathbf{h}^{l} \in \mathbb{R}^{M}$ is the input of the layer $l+1$, $\mathbf{W}^{l+1} \in \mathbb{R}^{MxN}$ and $\mathbf{b}^{l+1} \in \mathbb{R}^{N}$, the weight matrix and bias vector respectively, are the learnable parameters of layer $l+1$, σ is the activation function and finally $\mathbf{h}^{l+1} \in \mathbb{R}^{N}$ is the output of layer $l+1$.

A key aspect of deep learning is the ability of the network to learn from data. This learning process is performed by adjusting the weights and biases of the neurons such that the output of the network closely matches the desired output. This process is known as training and, specifically during the back propagation, the error between the desired output and the prediction of the model is propagated back through the network, and the parameters are adjusted to minimize the error.

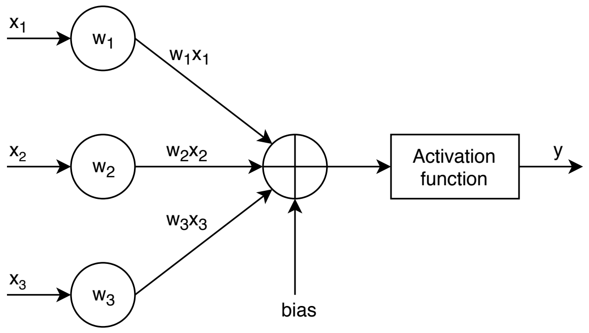


Fig. 2.1 Example of a single neuron with three input signals.

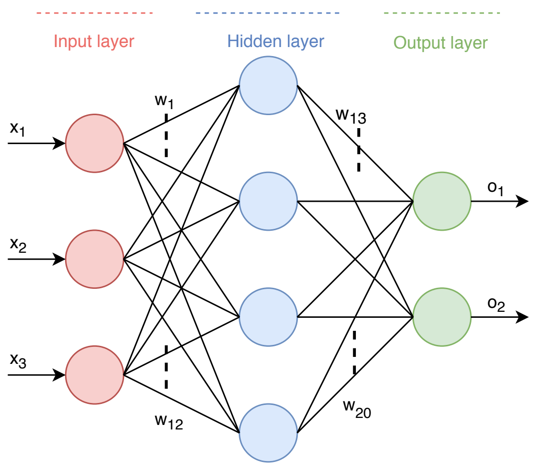


Fig. 2.2 Example of a fully connected neural network with one hidden layer.

Activation function

The activation function is used to add non-linearity to the neural network, and it is applied to the outputs of a neuron. There are many different types of activation functions, each with their own unique properties. The most common activation functions used in deep learning are:

- the sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$
- the hyperbolic tangent function: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- the ReLU function: $f(x) = \max(0, x)$
- the leaky ReLU function: $f(x) = \max(\alpha * x, x)$, where α is a small value, typically 0.01

The choice of activation function can have a significant impact on the performance of the network, and different activation functions are typically used for different types of tasks.

2.1.2 Convolutional Neural Networks

A convolutional neural network (CNN) is a type of deep learning model that is particularly well-suited for processing data with a grid-like structure, such images and videos. CNNs are able to take into account the spatial structure of the input data thanks to the convolutional layer, the core building block of the network. This allows CNNs to learn spatial hierarchies of features, such as edges, textures, and shapes, from the input data. Furthermore, the convolutional layers help the network to learn a compact representation of the data that is more efficient and more robust to small translations and distortions of the input data. Moreover, CNNs use pooling layers to reduce the spatial dimensionality of the feature maps produced by the convolutional layer, allowing the network to learn features at different scales.

Convolutional layer

The convolutional layer is the core of a CNN. It performs a mathematical operation called convolution, which is a way of combining input data with a set of learnable weights, also called filters, to produce a new set of features, called feature map. A set of filters are learned by the network during the training process. Each filter is a small matrix of weights of generic dimension (K,L) that is applied to a small region of the input data, known as a receptive field, to produce a single value in the output feature map. The operation can be defined as follow:

$$\mathbf{h}_{i,j}^{l+1} = \sum_{k=0}^{K}\sum_{l=0}^{L} \mathbf{h}_{i+k,j+l} * filter(k,l), \tag{2.2}$$

where $\mathbf{h}_{i,j}$ is the value of the input data at position (i,j), $filter(k,l)$ is the value of the filter at position (k,l) and $\mathbf{h}_{i,j}^{l+1}$ is the value of the output feature map at position (i,j).

At high level the operation can be seen a sliding window over the input matrix: the filters slides over the input isolating windows of data, and linearly combines the

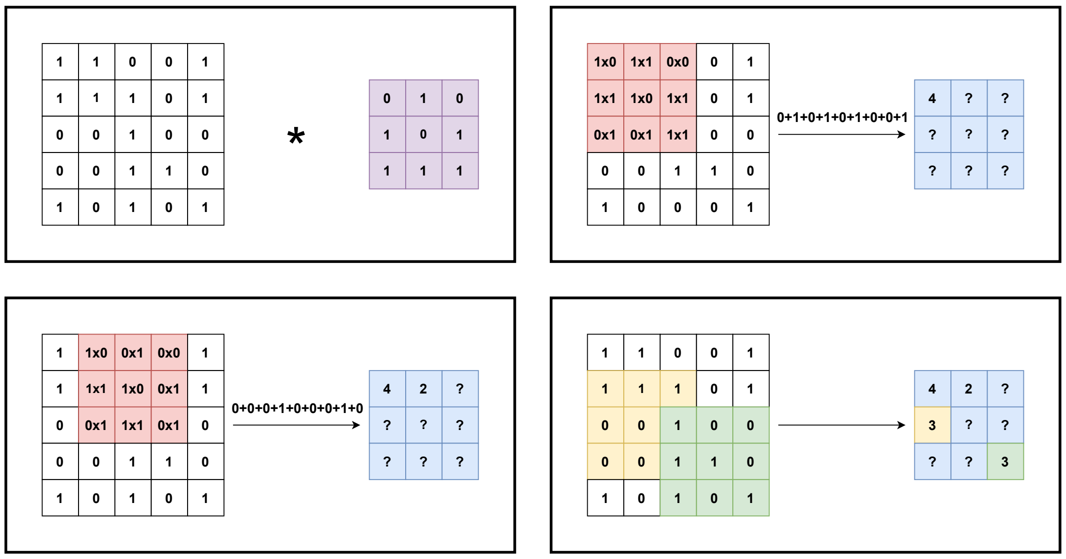


Fig. 2.3 Example of a convolution operation. 5x5 input data convolved with a 3x3 filter (top left). Computation of the element (1,1) (top right), of the element (1,2) (bottom left), of elements (2,1) and (3,3) (bottom right).

selected elements with the filter values. See Fig. 2.3 for an example of convolution operation.

Pooling layer

The pooling layer is another key component of CNNs. It is used to reduce the spatial dimensions of the feature maps produced by the convolutional layers and to introduce some degree of translation invariance. Furthermore, it is used to enlarge the receptive filed, i.e. the region of the input that each neuron is able to "see" and interact with. There are several types of pooling layers, the most common ones are max-pooling and average-pooling. Max-pooling selects the maximum value from a small region of the feature map, known as a pooling window or kernel, and uses it as the output value for the corresponding position in the pooled feature map. Average-pooling, on the other hand, computes the average value of the elements in the pooling window and uses it as the output value for the corresponding position in the pooled feature map.

2.2 Graph Convolutional Neural Networks

In the recent years, deep neural networks have obtained astonishing results on a large variety of problems, especially for image processing tasks, thanks to the power of convolutional neural networks. Although, the classic convolution operation can only be applied on data characterized by a regular structure. Therefore, it became tremendously interesting to formulate a general definition of a convolution operation suitable for any type of signal. In particular, there has been a growing attention to the not straightforward problem of extending CNNs to signals that can not be represented by regular structures or lie on non-Euclidean domains but can be easily defined on graphs.

A graph convolution operation can be formulated over different domains, and according to the literature two main approaches can be identified. Several methods [22–24] exploit the frequency domain and the graph Fourier transform. To limit the high computational costs, polynomial approximations [23] have been introduced. Among those methods, the most noticeable one is the Graph Convolutional Network (GCN) [24], for semi-supervised problems, such semi-supervised node classification or semi-supervised multi class classification. To clarify, semi-supervised node classification is defined as training a neural network with a training dataset only partially labeled to perform node classification uncovering similarities between the nodes of the graph. However, a critical limitation of this formulation is the inability to generalize the learned filters, computed over the spectrum of the graph Laplacian, to a variable graph structure.
The second class of approaches defines the graph-convolution operation in the spatial domain. In this scenario the graph convolution is defined as a local, i.e. computed over a neighborhood, weighted aggregation of signals. Since it is defined at neighborhood level, this formulation is suitable for any type of signal that can be defined over a graph, even with a variable graph structure, solving one of the crucial limitations of spectral approaches. Several definitions are present in the literature [25–33]. One important difference among those methods is the computation of the weights used in the aggregation. Many [25, 26, 29] use scalar weights or matrices that do not depend from the input data [27, 30, 31, 33]. Instead [28] proposes edge-dependent matrices as weights, leading to build an operation with more representational power. For the above reasons a special interest is given to the Edge-Conditioned Convolution (ECC) [28]

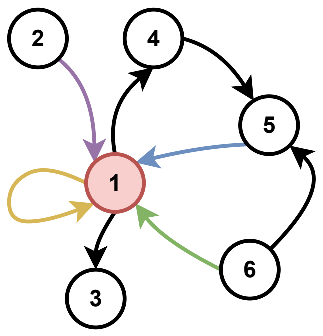


Fig. 2.4 Example of graph aggregation. The neighborhood of the node 1 is shown and the considered edges are highlighted with colors.

2.2.1 Edge-Conditioned Convolution

In this section the Edge-Conditioned Convolution [28], a powerful spatial graph convolutions, is described.

We consider a generic graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$, with n labeled vertices and m labeled edges. The ECC is implemented as a feed-forward neural network with l_{max} number of layers and the labeling functions $H^l : \mathscr{V} \to \mathbb{R}^{d_l}$, where d_l is the dimension of the feature vector at layer l, that associates at each vertex a feature vector, and $L : \mathscr{E} \to \mathbb{R}^s$, where s is the dimension of the attributes, that associates to each edge an attribute.

As previously stated, the ECC is defined as a local weighted aggregation of points, therefore three different aspects play an important role: the locality, i.e. the neighborhood, the computation of the weights and the type of aggregation.

The neighborhood $N^l(i)$ related to the node i at layer l is defined as the collection of the adjacent nodes and the node i itself:

$$N^l(i) = \{j;\ (j,i) \in \mathscr{E}\} \cup \{i\}. \tag{2.3}$$

The aggregation for each point employed in the operation is done at neighborhood level, so just among the signals that belong to their neighborhood. See Fig. 2.4 as an example of neighborhood. The ECC layer operation computes the feature vector $h^{l+1}(i)$ related to the i^{th} vertex as a weighted summation of signals $h^l(j) \in N^l(i)$ belonging to its neighborhood, therefore it can be applied to different structures of graphs without any constraints.

Another topic worthy of discussion is the delineation of the weights matrix. We consider a filter-generating network $F^l_{\mathbf{w}^l} : \mathbb{R}^s \rightarrow \mathbb{R}^{d_{l+1} \times d_l}$, implemented as a multilayer perceptron with two layers and $\mathbf{w}^l$ learnable weights. This function takes as input the label associated to each edge $L(j,i)$ and estimates the matrix $\mathbf{\Theta}^l_{ji}$:

$$\mathbf{\Theta}^l_{ji} = F^l_{\mathbf{w}^l}(L(j,i)) \in \mathbb{R}^{d_{l+1} \times d_l} \tag{2.4}$$

The edge labeling function considered is simply the difference between the features associated to the two nodes that the edge connects together:

$$L(j,i) = \mathbf{h}^l_j - \mathbf{h}^l_i,\ j \in N(i)^l, \tag{2.5}$$

where the vector $\mathbf{h}^l_i$ is the feature vector of node i at l^{th} layer. Therefore the weight matrix is computed as:

$$\mathbf{\Theta}^l_{ji} = F^l_{\mathbf{w}^l}(\mathbf{h}^l_j - \mathbf{h}^l_i) \tag{2.6}$$

Finally the aggregation chosen is a summation.

In conclusion, the final equation that describes the ECC is defined as:

$$\mathbf{h}^{l+1}_i = \frac{1}{|N(i)^l|}\sigma\left(\sum_{j \in N(i)^l} \mathbf{\Theta}^l_{ji}\mathbf{h}^l_j + b^l\right), \tag{2.7}$$

where σ is the activation function of the neural network, typically a ReLu function.

2.2.2 Lightweight Edge-Conditioned Convolution

In this section the lightweight Edge-Conditioned Convolution [34], a modified version of the original Edge-Conditioned Convolution [28] conceived in the context of image denoising, is described.

The lightweight ECC introduces some approximations to address the problems of vanishing gradients, over-parameterization and computation cost. The Lightweight

ECC is defined as follow:

$$\begin{aligned}\mathbf{h}_i^{l+1} &= \mathbf{W}^l \mathbf{h}_i^l + \sum_{j \in N(i)^l} \gamma^{l,j\to i} \frac{\boldsymbol{\Theta}_{ji}^l \mathbf{h}_j^l}{|N(i)^l|} \\ &= \mathbf{W}^l \mathbf{h}_i^l + \sum_{j \in N(i)^l} \gamma^{l,j\to i} \frac{\sum_{t=1}^r \omega_t^{j\to i} \boldsymbol{\phi}_t^{j\to i} \boldsymbol{\psi}_t^{j\to i^T} \mathbf{h}_j^l}{|N(i)^l|}. \end{aligned} \tag{2.8}$$

The local weighted summation is divided into two components: a self-loop part, with matrix $\mathbf{W}^l \in \mathbb{R}^{d_{l+1} \times d_l}$ which is shared among all points, and a neighborhood part, with weight matrix $\boldsymbol{\Theta}_{ji}^l$ that is defined by vectors $\boldsymbol{\phi}_t^{j\to i} \in \mathbb{R}^{d_{l+1}}$, $\boldsymbol{\psi}_t^{j\to i} \in \mathbb{R}^{d_l}$ and scalar $\omega_t^{j\to i}$. The parameter $\gamma^{l,j\to i}$ is an edge attention term which exponentially depends on the Euclidean distance between feature vectors:

$$\gamma^{l,j\to i} = \exp\left(-\|\mathbf{h}_i^l - \mathbf{h}_j^l\|_2^2/\delta\right), \tag{2.9}$$

where δ is a decay hyperparameter. This term takes inspiration from the edge attention mechanism introduced in [35] with the purpose of stabilizing the training procedure by giving less importance to edges between nodes with distant feature vectors.

The weights of the neighborhood aggregation play a crucial role in the original definition of ECC. They are implemented through a two layer fully connected network $F_{\mathbf{w}^l}^l : \mathbb{R}^{d_l} \to \mathbb{R}^{d_{l+1} \times d_l}$, a powerful definition that unfortunately brings some limitations.
First of all, there is the risk of over-parameterization: assuming $d_{l+1} = d_l$, the number of weights has a cubic dependence on the number of features. This tremendous amount of parameters may lead to vanishing gradients and overfitting. To overcome this problem a circulant approximation is proposed. The weight matrix is forced to be composed by circulant matrices, square matrices where all the rows have the same elements shifted to the right by one position with respect to the previous combination. In this way the only learnable parameters are the elements of the first row of each matrix, leading to a decrement of parameters by a factor equal to the number of possible shifts m.
Furthermore, a non-negligible aspect is the computation complexity related to the weights matrices. Infact, a matrix $\boldsymbol{\Theta}_{ji}^l$ has to be computed for each node j in the neighborhood $N(i)$ of each node i present in the signal. The amount of computation

required can quickly scale with the amount of input signal, leading to prohibitive memory requirements. Therefore, to solve this issue a low-rank approximation is introduced. According to the singular value decomposition of a matrix, a low rank approximation of rank r can be obtained by keeping the top, i.e. largest, r singular values:

$$\boldsymbol{A} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Psi}^T = \sum_i \lambda_i \boldsymbol{\phi}_i \boldsymbol{\psi}_i^T \approx \sum_{i=1}^{r} \lambda_i \boldsymbol{\phi}_i \boldsymbol{\psi}_i^T \tag{2.10}$$

where $\boldsymbol{\Phi}$ is a m x m orthogonal matrix and its columns $\boldsymbol{\phi}_i$ are the left singular vectors of $\boldsymbol{A}$; $\boldsymbol{\Psi}$ is a n x n orthogonal matrix and its columns $\boldsymbol{\psi}_i$ are the right singular vectors of $\boldsymbol{A}$; $\boldsymbol{\Lambda}$ is a m x n diagonal matrix and its non-negative entries λ_i are the singular values of $\boldsymbol{A}$. Therefore the weight matrix can be defined:

$$\boldsymbol{\Theta}^l_{ji} = \sum_{t=1}^{r} \omega_t \boldsymbol{\phi}_t^{j\to i} \boldsymbol{\psi}_t^{j\to i^T}, \tag{2.11}$$

where $\boldsymbol{\phi}_t^{j\to i} \in \mathbb{R}^{d_l}$, $\boldsymbol{\psi}_t^{j\to i} \in \mathbb{R}^{d_{l+1}}$ and ω_t is a scalar.
Following this new definition the $F^l_{w^l}$ network provide as output the components ω_t, $\boldsymbol{\phi}_t^{j\to i}$, and $\boldsymbol{\psi}_t^{j\to i}$ in three parallel branches. This novel implementation drastically reduce the memory occupation, the computational cost and the number of parameters.

2.3 Implicit Neural Representations

Recently, in computer vision and graphics, the new research field of neural implicit representations has emerged. There has been a growing attention in finding alternatives to the traditional discrete signal representations and continuous implicit functions parameterized by neural networks arose as a promising solution.

To parameterize a continuous function a simple fully-connected neural network, i.e. a multilayer perceptron (MLP), takes a low-dimension coordinated-based input and provides as output the desired function. Some example of applications, specifically 2-D image and 3-D shape regression, are analyzed to better understand the paradigm. In the 2-D scenario, the network learns to map the pixel coordinates of an image (x,y) to their corresponding rgb values (r,g,b). Instead, for 3-D shape regression, the model learns the 3-D representation of a point cloud taking as input a generic 3D coordinate (x,y,z) and learn to map it to a binary occupancy information (0 if the point is outside of the shape, 1 otherwise) [36]. Neural implicit representa-

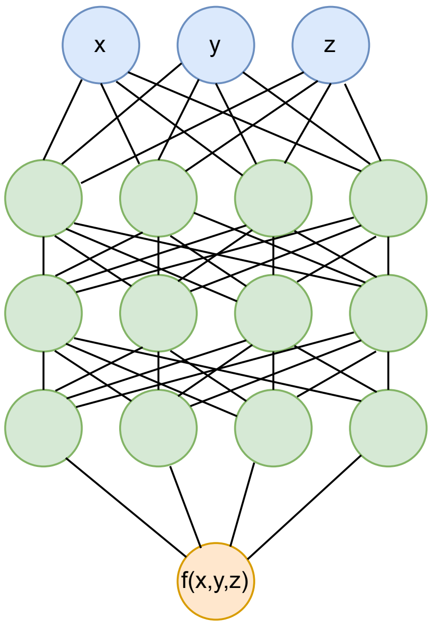


Fig. 2.5 Generic implicit neural representation. The input signal (x,y,z) is processed by a MLP with 3 hidden layers and the results of the function $f(x,y,z)$ is provided as output.

tions have been widely used to address 3-D processing tasks, such to represent signed distance [37, 38], occupancy fields [39, 36] or texture fields [40]. Furthermore, they are able to achieve state-of-the-art results on a variety of tasks, such for 3-D shape representation, texture synthesis and view synthesis[41].

However, the biggest limitation of such networks, when not applied to 3-D processing task, is their incapacity of modeling signals with fine details. This is partially cause by the utilization of ReLU as activation functions. Because of the definition of such functions, their second derivative is always zero, consequently they are not able to model information from higher-order derivatives of signals, especially in the case of natural images. Therefore, the simple structure of standard MLPs is not well suited to learn high-frequency functions, and even the task of fitting a real-world image is challenging. Recently, few works [41–43] have started to investigate the importance of sinusoidal mapping and following researches have proven periodic activation functions [1] or Fourier features embedding layers [44] as a key ingredient to successfully represent high-frequency content. Throughout this dissertation, a particular attention is given to the work of Siltzmann et al [1] and their proposed implicit neural representations based on sinusoidal functions.

2.3.1 Implicit Neural Representations with Period Activation Functions

The use of periodic activation functions for implicit neural representations has been recently proposed in [1] and their ability to represent complex natural signals and their derivatives is demonstrated.

Implicit neural representations have showed promising results in several applications, although they are not able to learn high-frequency signal component, as described in Sec. 2.3. Moreover, they are not capable of representing the derivatives of the desired signal, due to the utilization of ReLU activation functions in the networks. The second derivative associated to this activation function is always null, preventing the model to learn information contained in higher-order derivatives. To overcome the aforementioned limitations, a fully-connected network with periodic activation functions, called SIREN, is proposed:

$$\Phi(x) = \mathbf{W}_n(\phi_{n-1} \circ \phi_{n-2} \circ ... \circ \phi_0)(\mathbf{x}) + \mathbf{b}_n, \tag{2.12}$$

$$\mathbf{x}_i \to \phi_i(\mathbf{x}_i) = \sin(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i) \tag{2.13}$$

where $\phi_i : \mathbb{R}^{M_i} \to \mathbb{R}^{N}_{i}$ is the i^{th} fully-connected layer, with weights $\mathbf{W}_i \in \mathbb{R}^{N_i \times M_i}$ and biases $\mathbf{b}_i \in \mathbb{R}^{N_i}$ and $\mathbf{x}_i \in \mathbb{R}^{M_i}$ is the coordinate input.

It is interesting to observe that the derivative of a SIREN, i.e. a MLP with sine as activation function, is still a SIREN, because the derivative of a sine function is a cosine, that can be seen as a shifted sine. Therefore, with little supervision the network is able to successfully learn the derivative of a signal.

Consider as a simple application the task of image fitting. The task can be defined as learning the continuous function $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$ that maps the pixel coordinates $\mathbf{x}_i = (x_i, y_i)$ to their corresponding RGB values $f(\mathbf{x}_i)$. The network is trained with the Mean Squared Error (MSE) loss function between the reconstructed values $\Phi(\mathbf{x}_i)$ and the real values $f(\mathbf{x}_i)$:

$$L = \sum_i ||\Phi(\mathbf{x}_i) - f(\mathbf{x}_i)||^2 \tag{2.14}$$

In Fig. 2.6 it is possible to see the results obtained by the SIREN network compared to other architectures with different activation functions. First, it is possible to notice

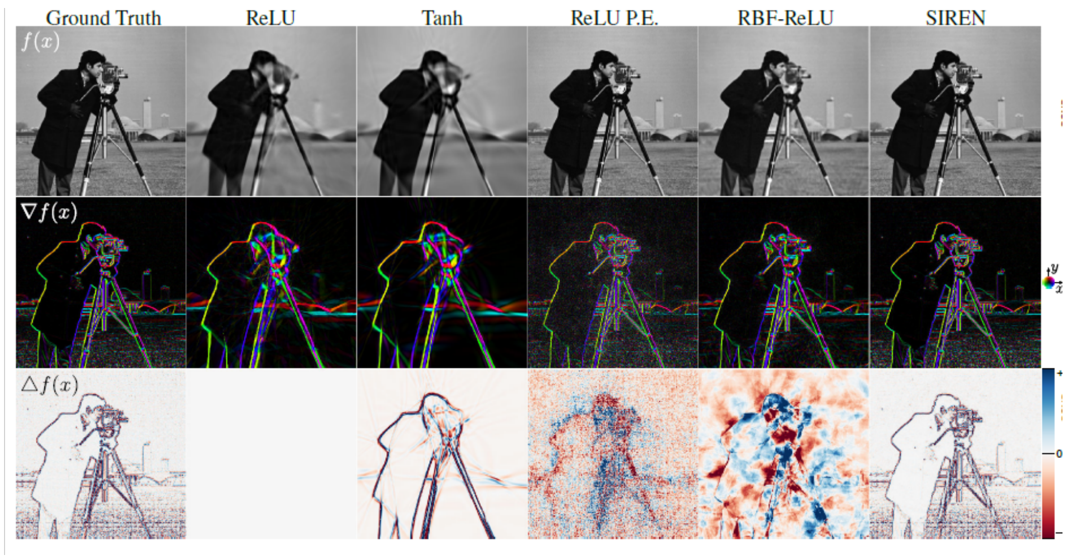


Fig. 2.6 Comparison of several comparable implicit network architectures with different activation functions (from left to right: ReLU, Tanh, ReLU with Positional Encoding (P.E.), Radial basis function (RBF) with ReLU and SIREN) trained for the task of natural image fitting (top left: ground truth image). The second and the third rows show first- and second-order derivatives of the function fit. Image taken from [1] paper.

that only SIREN and ReLU P.E. [41], a ReLu network with a sinusoidal positional encoding, are able to efficiently reconstruct the desired image. Moreover, SIREN network is the only one able to reconstruct the second derivative of the signal (last row of Fig. 2.6). One crucial point to obtain an effective training is the initialization scheme. The input of each sinusoidal activation has to be normally distributed with a unitary standard deviation in order to keep stable the distribution of the output of the activation among the network and obtain high accuracy results and good speed of convergence.

Recalling that given an input $\mathbf{x} \sim U(-1,1)$, i.e. uniformly distributed between -1 and 1, the output of a simple layer with sinusoidal activation function, defined as $y = \sin(\mathbf{w}^T\mathbf{x} + b)$, has an acsine distribution $y \sim \arcsin(-1,1)$. It can be demonstrated that having a weight matrix uniformly distributed keep the distribution of subsequent SIREN layer acsine distributed.

Overall, this research contributed to novel discoveries in the field of neural implicit representations and proved that the proposed SIRENs are able to obtain remarkable results in a large variety of tasks, going from the simple image fitting to solving Poisson equations.

2.4 Transform Coding

Transform coding is a technique used in digital signal processing and in data compression to represent a signal in a different domain. The general idea is to represent the signal in a domain where the majority of the information is stored in only few elements or coefficients, which can be quantized, encoded and saved or transmitted more efficiently.

Given an input sequence x_n divided into N blocks, a reversible function is chosen to map each block into a transform sequence y_n. After applying the transform function, it can be observed that different elements have different statistical characteristics, e.g. magnitude, which reflect the signal information stored in those elements. The transformed elements are then quantized taking into account the different statistics and the data loss. Finally the quantized elements are encoded with an entropy coding technique, such Huffman or arithmetic coding.

The considered transforms are linear and reversible, and for 1-D data sequence they can be represented as:

$$y_n = \sum_{i=0}^{N-1} x_i a_{n,i}, \tag{2.15}$$

$$x_n = \sum_{i=0}^{N-1} y_i b_{n,i}, \tag{2.16}$$

where y_n and x_n are the output and input sequence and sequence and a_n and b_n are the transform coefficients. Another possible representation is in the matrix form:

$$\mathbf{y} = \mathbf{A}\mathbf{x}, \tag{2.17}$$

$$\mathbf{x} = \mathbf{B}\mathbf{y}, \tag{2.18}$$

where $\mathbf{A}$ and $\mathbf{B}$ are $N \times N$ matrices where each element is a transform coefficient.

Transform coding is typically used for image compression, therefore particular attention is dedicated to two dimensional transforms. Given an input image $X_{i,j}$, where the indices (i,j) refer to the $(i,j)^{th}$ pixel, a generic 2-D transform can be defined for a NxN block size:

$$\mathbf{Y} = \mathbf{A}\mathbf{X}\mathbf{A}^T, \tag{2.19}$$

$$\mathbf{X} = \mathbf{B}\mathbf{Y}\mathbf{B}^T. \tag{2.20}$$

It is important to notice that such 2-D transform is separable and orthonormal. Since the transform is separable, it can be first computed the transform along one dimension (e.g rows) and then to the other (e.g. columns). Moreover, orthonormal matrices have their inverse matrix equal to their transpose, and Eq. 2.20 can be re-written as:

$$\mathbf{X} = \mathbf{A}^T\mathbf{Y}\mathbf{A}. \tag{2.21}$$

Therefore, orthonormal transforms are energy preserving, or lossless.
In the following sections a well-known transform commonly used in transform coding, the Discrete Cosine Transform, and the quantization stage are better described. Finally a representative example of a standard image compression technique is reported.

2.4.1 Discrete Cosine Transform

The most common 2-D transform used in image compression is the Discrete Cosine Transform (DCT), that converts data from spatial to frequency domain.

The DCT is derived from the Discrete Fourier Transform (DFT), and based on cosines functions. The N-point 1-D DCT is defined as:

$$\hat{y}_k = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} \Lambda(k) y_n \cos\left(\frac{k\pi}{N}\left(n + \frac{1}{2}\right)\right) \tag{2.22}$$

and the corresponding inverse form is

$$y_n = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} \Lambda(k) \hat{y}_k \cos\left(\frac{k\pi}{N}\left(n + \frac{1}{2}\right)\right) \tag{2.23}$$

where

$$\Lambda(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if k=0} \\ 1 & \text{otherwise} \end{cases} \tag{2.24}$$

Therefore the orthonormal DCT basis functions are

$$\theta_{k,n} = \begin{cases} \sqrt{\frac{1}{N}} & \text{if k=0} \\ \\ \sqrt{\frac{2}{N}} \cos\left(\frac{k\pi}{N}\left(n+\frac{1}{2}\right)\right) & \text{otherwise} \end{cases} \tag{2.25}$$

One of the most interesting property of the DCT is that it separates the image into a set of frequency components such that the low-frequency are concentrated in the upper-left corner of the transform and the high-frequency in the lower-right corner. This compact distribution is exploited to obtain an efficient compressed signal representation simply removing or reducing the high-frequency components. Notably, the DCT is the core of the JPEG standard for image compression.

2.4.2 Quantization

In transform coding, quantization is the process of mapping a continuous range of values, obtained after the transform, to a discrete set of values. This step is used to reduce the number of bits needed to represent the data, while still preserving enough information to accurately reconstruct the original signal.

To enhance the compression efficiency it is important to find algorithms able to assign different number of bits according to the amount of information stored by each transform coefficients. Some approaches are based on known properties of the transform, and perform the bit allocation according to estimated variances of the coefficients, where at higher variance correspond a higher number of bits. This is a very simple algorithm, able to provide good results. However, it is not robust to outlier, since local variations are lost considering just the average value. To avoid this issue it is introduced the threshold coding, where the coefficients are quantized and saved are chosen according to a specific threshold.

2.4.3 Image Compression Standard

The algorithm developed by the Joint Photographic Experts Group (JPEG) is the most well-known standard for lossy image compression.

The transform used in JPEG is the DCT described in Sec. 2.4.1. The image is divided into 8×8 blocks, each transformed following Eq.2.22.

For instance, the first 8×8 block of Lena image, converted in Ybc format, is:

$$\begin{bmatrix} 162 & 162 & 162 & 161 & 162 & 157 & 163 & 161 \\ 162 & 162 & 162 & 161 & 162 & 157 & 163 & 161 \\ 162 & 162 & 162 & 161 & 162 & 157 & 163 & 161 \\ 162 & 162 & 162 & 161 & 162 & 157 & 163 & 161 \\ 162 & 162 & 162 & 161 & 162 & 157 & 163 & 161 \\ 164 & 164 & 158 & 155 & 161 & 159 & 159 & 160 \\ 160 & 160 & 163 & 158 & 160 & 162 & 159 & 156 \\ 159 & 159 & 155 & 157 & 158 & 159 & 156 & 157 \end{bmatrix}$$

The corresponding DCT coeffients are:

$$10^3 * \begin{bmatrix} 1.2838 & 0.0058 & 0.0029 & -0.0003 & 0.0003 & -0.0003 & -0.0054 & 0.0066 \\ 0.0082 & -0.0003 & 0.0007 & -0.0050 & 0.0019 & 0.0032 & -0.0041 & 0.0034 \\ -0.0052 & -0.0006 & -0.0016 & 0.0015 & -0.0007 & -0.0006 & 0.0020 & -0.0019 \\ 0.0024 & 0.0013 & 0.0016 & 0.0013 & -0.0007 & -0.0013 & 0.0002 & 0.0005 \\ -0.0011 & -0.0014 & -0.0003 & -0.0017 & 0.0017 & 0.0010 & -0.0014 & -0.0001 \\ 0.0014 & 0.0008 & -0.0017 & -0.0000 & -0.0021 & 0.0008 & 0.0015 & 0.0007 \\ -0.0019 & -0.0001 & 0.0029 & 0.0017 & 0.0017 & -0.0024 & -0.0009 & -0.0012 \\ 0.0015 & -0.0001 & -0.0023 & -0.0017 & -0.0010 & 0.0021 & 0.0004 & 0.0009 \end{bmatrix}$$

After the transform, each block is quantized, following a uniform mid-tread quantization:

$$l_{i,j} = \left\lfloor \frac{\hat{y}_{i.j}}{Q_{i,j}} + 0.5 \right\rfloor, \tag{2.26}$$

where $\hat{y}_{i,j}$ is the transform coefficient at the i^{th} row and j^{th} column and $Q_{i,j}$ is the corresponding quantization step. The quantization step values are taken from a fixed look-at-table. Example of quantization table for the luminance part:

$$\begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99
\end{bmatrix}$$

Analyzing the quantization look-at-table it is possible to observe that the quantization step increases moving from the top-left to the bottom-right of the matrix. Recalling that on the top-left part of the DCT coefficient matrix are stored the low-frequency coefficients, a higher step size is applied to higher order coefficients. Consequently, more quantization error is introduced to high-frequency (AC) coefficients rather than to the low-frequency (DC) ones. This design decision is based on the perception from the human visual system. In particular, humans are more sensible to errors of low frequency coefficients.

Finally the quantized coefficients are encoded. JPEG utilizes two different type of encoding for DC and AC coefficients, based on Huffman coding and specifically fine tuned for this specific compression algorithm. The details of such encoding mechanism are out of the scope of this brief introduction on the JPEG algorithm.

Overall, the JPEG standard provides a way to compress digital images while maintaining a good balance between file size and image quality.

2.5 Point Cloud Processing Tasks

In this section some general aspects of the problems addressed in this dissertation are analyzed. In particular, the type of data involved along with the specific processing tasks are described.

2.5.1 Point Clouds

A point cloud is an unordered set of 3-D points that represents the geometry of an object or an entire scene. Point clouds can be acquired by various types of sensors, such as cameras, range scanners, or laser scanners. The obtained point cloud is usually represented as a set of discrete points, which can be stored in a variety of formats, such as ASCII, binary, or PLY. One of the most common method for acquiring point clouds is via LIDAR (Light Detection and Ranging) sensors. To obtain the 3-D location of the points, the distance between the sensor and the point is computed. This type of sensor emits laser pulses and measure the time it takes to bounce back. This method allows a high precise, long-range and fast acquisition, making it suitable for outdoor environments and mobile applications.

Another popular method exploits stereo cameras, a system of two or more cameras that capture images of the same scene from slightly different viewpoints. The stereo correspondence problem, i.e. finding corresponding pixels in different images, is used to retrieve the 3-D coordinates of the points. This method is suitable for indoor environments, and it can produce both dense and sparse point clouds, depending on the resolution and the quality of the cameras.

A third common method uses time-of-flight (ToF) cameras, such devices measure the time it takes for a light pulse to travel from the camera to the point and back. ToF cameras can be used in both indoor and outdoor environments and can produce dense point clouds, but they are highly sensitive to the lighting conditions and can suffer from motion blur.

Finally, structured light systems can be used. They project a pattern of lights onto the scene and capture the deformed pattern using a camera. The correlation between the projected pattern and the captured image is used to calculate the 3-D coordinates of the points. This method can produce dense point clouds with high precision, but it is sensitive to the texture and reflectance of the scene.

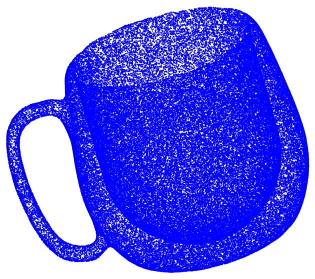
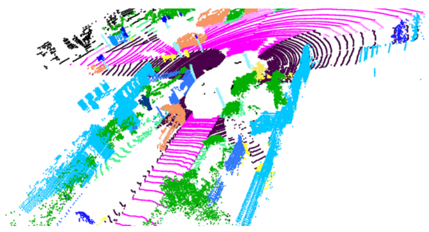



Fig. 2.7 Example of well-known point clouds. Cup from Shapenet dataset [2] (left) and outdoor lidar street with multiple objects from SematicKITTI dataset [3] (right).

Thanks to the great availability of instruments able to capture this type of data and their ability to provide an accurate representation of real world scenarios, point clouds are becoming increasingly popular. Overall, point clouds have many applications in autonomous driving, robotics, 3-D sensing, and medical imaging, such as 3-D modeling, surface reconstruction, object recognition, scene understanding, and navigation. In addition, point clouds can be used to perform various types of geometric and photometric processing tasks, such as surface normals estimation, surface smoothing, and texture mapping. Therefore, recently there has been a growing interest in founding always better methods to address point cloud processing tasks.

2.5.2 Point Cloud Denoising

The great availability of instruments, such LIDARs, able to collect point clouds is one of the fundamental reasons of the growing interest and utilization of this type of data. However, all the acquisition methods are imperfect and perturb the data adding an important amount of noise and outliers to the original shape. This data disruption negatively affect the performance of several downstream tasks, such as surface reconstruction, object segmentation or normal estimation. Therefore, it is of paramount importance to tackle the task of point cloud denoising.

In the literature several approaches can be found and they can be roughly divided in: local surface fitting methods [45–50], sparsity-based methods [51–53], graph-based methods [54–56], and learning-based methods [57–61]. The first three categories are traditional methods, i.e. optimization-based. Among the first category there are methods quite common, like Moving Least Square [45], that progressively fit the surface upon the noisy data. They are able to obtain good performance when the amount of noise is limited, otherwise they often suffer of oversmoothing. Instead, sparsity-based methods are focused on first estimating the surface normals for each point and then adjusting the point position accordingly. The performance of such methods are inevitably linked to the ability to estimate the normals, that is limited at high level of noise, worsening the overall results. The last traditional-based category has its origin in the graph theory. Those methods build a graph upon the noisy data and then apply graph-total variation based regularization for the restoration task. Those methods are very effective at low level of noise, although at high level the performance are degraded by the unstable graph construction.

Recently, learning-based approaches have gained attention due to the remarkable results obtained by deep neural networks, and especially by convolutional neural networks in restoration tasks, such as image denoising. Although, the extension of such techniques to point clouds is not trivial, mostly due to the fact that such data lie on irregular domains and are permutation invariant. One of the first neural network able to efficiently deal with point clouds is PointNet [8] conceived to tackle classification and segmentation tasks. PointNet proposes a network that processes each point independently and then apply a global aggregation. PointCleanNet [57] is an extension of this work to the task of point cloud denoising and outlier removal. In this work, two different networks are proposed, one focused on the task of outlier removal and another able to estimate the correction vectors for the noisy points. Other learned-based methods derived from PointNet use alternative approaches; [60] estimates the direction to the surface of the point clouds or [58] estimated the reference plane. Although they are able to fulfill the denoising task, all those methods inherited some limitations from the PointNet architecture. In particular, since they process each point independently, they are not able to exploit any of the desirable characteristics of convolutional neural networks, like the ability to build hierarchical features and to exploit local structures. PointCleanNet is able to overcome the last drawback, taking as input local patches and estimate the correction vector just for the center point, forcing to the network some local information.

Recently, in the context of point cloud classification and segmentation, graph-convolutional neural networks [62] have emerged as a novel and promising architecture. They are based on the graph convolution, an operation naturally able to deal with irregular domain and to exploit some of the desirable properties of traditional convolution. Therefore, such networks have been used to address some basic point cloud processing tasks such as classification [28], segmentation [31], shape completion [63] and generation [64]. In particular, DGCNN [31] is the first implementation that introduced the idea of a dynamic graph update in the hidden layers of a graph-convolutional network. However, the denoising problem is yet to be addresses since it is significantly different from the classification and segmentation tasks that rely more on global features instead of localized representations. Therefore, in this thesis a deep graph-convolutional neural network is proposed to denoise the point cloud geometry. The proposed architecture is characterized by a fully-convolutional behavior that, by design, can build hierarchies of local or non-local features to effectively regularize the denoising problem. The method is further extended to manage the presence of outliers, and a novel architecture able to simultaneously removes outliers and denoises the remaining points is proposed.

2.5.3 Point Cloud Normal Estimation

The task of point cloud surface normals estimation is a fundamental problem in the field of computer vision and 3-D sensing, which aims to estimate the orientation of the surface normals of a point cloud having as input just the 3-D coordinates of the input cloud. The surface normals of a point cloud are an essential information for many geometric and photometric processing tasks, such as surface reconstruction [65], shading [66], and denoising [55].

Principal Component Analysis (PCA) [67] is a technique widely used for dimensional reduction and feature extraction in various fields, and it can be used to estimate the surface normals of a point cloud. In this scenario, the PCA is applied to a set of local neighborhoods around each point, where the normal vectors are estimated based on the eigenvectors of the covariance matrix of the neighborhood points. Several extensions have been proposed with more accurate surface fitting techniques, such jet fitting [50], moving least squares [68], and spherical fitting [47]. However, PCA-based methods are sensitive to the density of the points and highly depend on the dimension of the chosen patch and on the level of noise.

Another class of approaches [69–71] is based on the Voronoi diagram. A Voronoi diagram of a point cloud provides a geometric representation of the space partitioned into regions of closest points. Such representation is exploited to estimate the local geometric structure of the point cloud and the surface normals of the points. These methods are robust and efficient techniques, particularly well suited for unstructured and noisy point cloud data. However, they have some intrinsic limitation, they highly rely on hyper-parameters and need pre-processing steps in presence of high levels of noise.

Recently, deep learning methods [72–76] has emerged as novel and promising direction. Although, as we already discusses in Sec. 2.5.2, it is not straightforward directly apply deep learning techniques to point clouds. One method is to change the original data structure of the point cloud in order to obtain a structured representation. In this way, standard convolutional neural networks can be used on the preprocessed data, at the expense of the some geometric infromation. Some methods [73, 75] adopt this approach, and exploit a local neighborhood to estimate the normal of the center point. Boulch et al. [75] use a Hough transform to obtain a 2-D representation of each neighborhoods, allowing to directly use 2-D CNNs at the cost of loosing data information. Instead, Nesti-Net [73] projects the local geometry on a grid with a multi-scale structure, techniques that improves the performance but drastically increase the computation time. Another noticeable approach is PCPNet [72], derived from the well-know PointNet [8] architecture. In this approach, each point of the selected patch is processed independently and after aggregate together in order to estimate the normal of the center point. Even if this approach does not introduce any data loss, it is not able to build hierarchical neighborhoods.

Thanks to the astonishing performance obtained by graph-convolutional neural networks in several tasks, they have progressively gained more attention. Therefore, in this thesis a graph-convolutional neural network is investigated to efficiently estimate the surface normals of point clouds.

2.5.4 Point Cloud Attribute Compression

Point cloud attribute compression is a task that aims to efficiently represent and transmit attributes, like color information, associated to each point of a point cloud. Point clouds are a geometric data type that recently gained popularity. They consist in a set of 3-D coordinates along with some attributes, like color information or surface normals associated to each point, provide additional visual information about the scene or object. Point clouds occupy a large amounts of storage space, making them difficult to process and transmit efficiently. Therefore, it is particularly important to develop models able to reduce their memory occupation.

Most of the methods available in the literature are focused on compressing 3-D point positions instead of their attributes. Nevertheless, these attributes are critical information to obtain high-quality rendered point clouds and have a large memory occupation. The Moving Picture Experts Group (MPEG) is the leader of standard point cloud compression (PCC) techniques. They propose two main approaches: Geometric-based PCC (G-PCC) and Video-based PCC (V-PCC). The first one is based on the original 3-D coordinates and data structure of the point clouds. Instead, the second approach project the 3-D data over a 2-D plane in order to use classic video compression algorithms, such HEVC.

Other geometry-based methods exploit graph transform [77–80] or the Region-Adaptive Hierarchical Transform (RAHT) [81], a method based on an hierarchical transform based on Haar wavelets. Among the different type of point cloud attributes, the point color information is extremely important for preserving the visual quality of point clouds and plays a big role in the data transmission and storage. Therefore, compression of color information can significantly reduce the data size and bandwidth requirements for transmitting point clouds over networks and also reduce the storage space required for point clouds. Moreover, it can improve the performance of 3-D point cloud visualization systems, especially on devices with limited computational resources.

Chapter 3

Learning Graph-Convolutional Representations for Point Cloud Denoising

3.1 Introduction

This study proposes a first deep graph-convolutional neural network that can effectively denoise point cloud geometry. The network has a fully-convolutional architecture that can construct hierarchies of local or non-local features to regularize the denoising problem. This is different from other methods that typically work on fixed-size patches or use global operations. The graph is dynamically computed from the high-dimensional feature-space representations of the points, allowing for the discovery of more complex latent correlations. Extensive experimental results show that this approach significantly outperforms state-of-the-art methods, particularly in high-noise conditions. It is also robust to structured noise distributions found in real LiDAR acquisitions.

The focus of the project is the investigation of the suitability of graph-convolutional neural networks to deal with point clouds and a first set of challenging scenarios, such as the presence of noise, is considered.

3.2 Proposed method

In this section the proposed Graph-convolutional Point Denoising Network (GPDNet), i.e., a deep neural network architecture to denoise the geometry of point clouds based on graph-convolutional layers, is presented.

3.2.1 Architecture

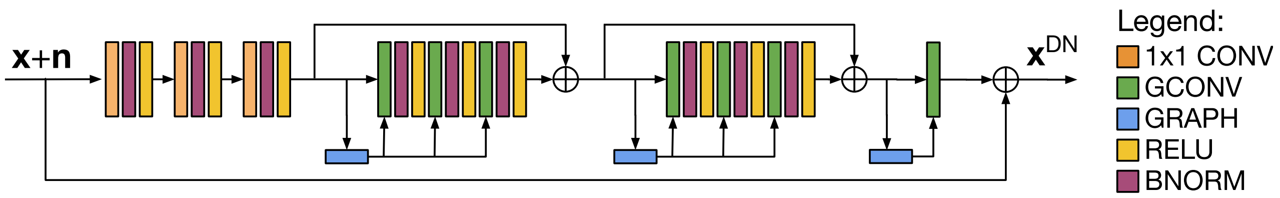


Fig. 3.1 GPDNet: graph-convolutional point cloud denoising network. The network takes as input the noisy point cloud $\mathbf{x}+\mathbf{n}$ and provides as output the denoised point cloud $\mathbf{x}^{DN}$. The network first projects the noisy point cloud into a feature space by means of 1-by-1 convolutions (1x1 CONV) and then with several graph convolutional layers (GCONV) it estimates the additive noise that is projected back into the 3-D space by a single graph-convolutional layer. The estimated noise is finally subtracted to the noisy input and the desired denoised point cloud is obtained.

The proposed GPDNet architecture is illustrated in Fig. 3.1. It can be seen that, at high level, It is a residual network that addresses the easier task of estimating the noise in the input point cloud, rather than directly cleaning the data. This design choice is justified by the fact that it has been shown [82] in the context of image denoising that estimating the residual is a much easier task than directly cleaning the data.

The network is divided into three blocks: point projection, noise estimation and output reconstruction. First, a module constituted by three single-point convolutions, each followed by a batch normalization layer and a Relu activation function, gradually project the 3-D points into an F-dimensional feature space. The first 1x1 convolutional layer maps the 3-D noisy input, $\mathbf{x} + \mathbf{n}$ in Fig. 3.1, into the feature space of dimension F_1, the second layer maps the features from dimension F_1 to F_2 and finally the third one to dimension F that will be kept for the rest of the network. After that, a cascade of two residual modules is inserted. Each residual block is constituted by three graph-convolutional layers, each followed by batch normalization and Relu activation function, and a skip connection to limit the vanishing gradient problem. All the operations performed in each residual block are in the F-dimensional space. The goal of each residual block is to gradually better estimate the additive noise component of the input, since the network exploit the residual learning strategy. Finally the additive noise predicted in the feature space by the residual blocks is projected back to the 3-D space by a final graph-convolutional layer and the denoised version of the point cloud can be obtained.

A key concept of the architecture is the graph construction. At the beginning of each residual block the graph is computed by selecting the k nearest neighbors to each point in terms of Euclidean distances in the feature space. This graph construction is called "dynamic", since it is updated throughout the network. In the proposed architecture, the dynamic graph construction is updated after each residual block, but shared among the graph-convolutional layers within each block to reduce the computational complexity. The dynamic graph construction is a powerful alternative to the most common fixed graph, where the graph is created at the beginning of the network using the original 3-D input data. In a dynamic graph construction the graph is computed over the similarities among the high-dimensional feature-space representations of the data points, which enables the discovery of more complex and subtle relationships within the data that cannot be found by analyzing the noisy 3-D space, and allows identifying non-local self-similarities. This particular graph construction has been shown [31, 64] to uncover the similarity between data and promote more powerful and meaningful feature representation. It is particularly beneficial in the context of a residual denoising network because it gradually reveals relationships that have not been previously identified. Moreover, this design choice is superior in presence of noisy input data, otherwise the graph built on top of the noisy data would be unstable or not optimal.

3.2.2 Graph-Convolutional layer

The graph-convolutional layer is the core of the proposed architecture. The Lightweight ECC [34], which is a modified version of the ECC presented in [28], is considered. The details of the operation are extensively described in Sec. 2.2.2.

The graph-convolutional layer is designed to have two inputs: a matrix $\mathbf{h}^{l+1} \in \mathbb{R}^{F^l \times N}$ that contains a feature vector for each of the N points in the point cloud, and a graph that describes the connections between the points. The output feature vectors $\mathbf{h}^{l+1} \in \mathbb{R}^{F^{l+1} \times N}$ at layer $l+1$ are calculated by combining the local information of the feature vectors according to the connections described in the graph. The combination is computed by weighting the local feature vectors based on the connections:

$$\mathbf{h}_i^{l+1} = \mathbf{W}^l \mathbf{h}_i^l + \sum_{j \in N(i)^l} \frac{\mathbf{\Theta}_{ji}^l \mathbf{h}_j^l}{|N(i)^l|},$$

where $\mathbf{h}_i^l$ is the input feature vector of node i at layer l, $N(i)^l$ is the set of its neighbors, $\mathbf{W}^l \in \mathbb{R}^{F^{l+1} \times F^l}$ is the self-loop matrix, $\mathbf{\Theta}_{ji}^l \in \mathbb{R}^{F^{l+1} \times F^l}$ is the neighbors weight matrix of node i containing the weights associated to each point j in the neighborhood of i, $N(i)^l$. To have more details on the computation of $\mathbf{\Theta}_{ji}^l$ see Sec. 2.2.2.

3.2.3 Loss functions

The proposed method has a supervised setting, i.e. the ground truth is available during the training phase, and two loss functions are considered.

First, we investigated the Mean Square Error (MSE), that compute a per point metric between the denoised point cloud $\hat{\mathbf{x}}$ and the noiseless version $\mathbf{x}$:

$$L_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} ||\hat{\mathbf{x}}_i - \mathbf{x}_i||_2^2, \tag{3.1}$$

being N the number of points in the point cloud. This is the most natural choice in presence of Gaussian noise.

However, in order to obtain a good denoised version of the points it is not necessary to recover their exact original position, but it is sufficient to move the point to the original surface. Therefore, the tangential component of the noise is not as relevant as the normal one. However, the MSE alone is not able to exploit this characteristic. To incorporate this property a regularization term that measures the distance between the denoised points and the ground truth surface can be inserted. This measure can be approximated by the proximity to surface metric which computes the distance between the denoised point and the closest ground truth point. In general, two point clouds can be considered close if every point of either cloud is close to some point of the other cloud. Therefore, to measure such closeness it is necessary to compute all the distances from each point of a cloud to some point, i.e. the closest, of the other, and vice-versa. In this specific scenario, it is interesting just half of such distance. Notably, PointCleanNet [57] already investigates and uses the measure of surface distance between clouds, although it employs both terms in the loss function. The addition of the second term, the distance between ground truth points from the denoised surface, is used to ensure that the denoised points do not collapse into filament structures. Although, in the proposed work the MSE has been experimentally found to enforce this property with better results. Therefore a novel loss function (MSE-SP) is defined as follow:

$$L_{\mathrm{MSE-SP}} = \frac{1}{N}\sum_{i=1}^{N}\left[||\hat{\mathbf{x}}_i - \mathbf{x}_i||_2^2 + \lambda \min_j ||\hat{\mathbf{x}}_i - \mathbf{x}_j||_2^2 \right] \tag{3.2}$$

with λ as the regularization hyperparameter. In Eq. 3.2.3 x_j is the nearest point with respect to the reconstructed x_i point in the ground truth point cloud. The proposed regularization is able to give more importance to the normal component of the distance because it optimize the points to be on the original surface, where more than one position have the same the normal component but different tangent one. In the literature other works already considered proximity to surface in the loss function, but it provoked some undesirable outputs, like clusterization of the points resulting in filament structures. PointCleanNet [57] to mitigate this side effect combines the surface proximity with a dual term measuring the distance between a ground truth point and the closest denoised point. Instead, we found that using the MSE to enforce this property provides better results.

3.3 Experimental results

In this section an experimental evaluation against state-of-the-art approaches as well as an analysis of the proposed technique is performed. In this section, experiments are conducted to evaluate the proposed technique and compare it with state-of-the-art approaches. Extensive analysis of the proposed technique are also reported.

3.3.1 Experimental setting

The training and testing data are obtained from the ShapeNet [2] database, which contains 3-D models of 55 object categories represented by a collection of meshes. To use this data, the points were sampled and normalized. Specifically, 30720 uniformly distributed points are sampled from each model, and the point clouds are scaled so that they are all the same size. More than 100000 patches, each containing 1024 points, are randomly selected from the point clouds to create the training set, which included point clouds from all categories except for ten reserved for the test set. The initial number of points, 30720, is chosen to potentially have 30 patches from each point cloud and ensure a sufficiently large a variegate training set. Each patch is created by randomly selecting a point and collecting its 1023 closest points. The test set consisted of 100 point clouds from ten different categories:: airplane, bench, car, chair, lamp, pillow, rifle, sofa, speaker, table. Ten models are randomly selected from each category, and 30720 uniformly distributed points sampled from each model. Note that since the training of GPDNet is done at patch level, any discrepancy between the number of points used in training and testing does not affect the performance. The network is able to generalize, and it proven by our experiments where input of 1024 points is used in training and 30720 points in testing. Instead, an aspect of interest is the point distribution since the network is quite sensible to its variation between training and testing data.

GPDNet was trained using a fixed noise variance for around 700000 iterations, with each iteration consisting of a batch size of 16. The number of features used in all layers was 99, except for the first three single-point convolutional layers, which had gradually increased numbers of features (33, 66, and 99). The Adam optimizer was used with a fixed learning rate of 10^{-4}. In terms of the graph-convolutional implementation, the rank was set to 11 for low-rank approximation, three circulant rows were used for constructing the circulant matrix, and a value of 10 was set for the graph-convolutional parameter δ. During testing, the entire point cloud was used as input and each point in the point cloud was associated with a search area in which neighbors were identified. By default, 16 nearest neighbors were used for graph construction unless stated otherwise.

3.3.2 Comparisons with state-of-the-art

In this section, the proposed approach is evaluated against state-of-the-art point cloud denoising methods. The literature review in Sec. 2.5.2 suggests that there are various categories of point cloud denoising methods, and at least one algorithm from each category is taken into account for the experiments. Specifically, the MLS-based surface fitting methods, APSS [47] and RIMLS [46], are tested using the MeshLab software [83] , while AWLOP [49] is another surface fitting method and MRPCA [53] is a sparsity-based method, both of which are implemented using the code provided by their respective authors. GLR [54], a graph-based method, is implemented using the authors' code, and PointCleanNet (PCN) [57], a recent learning-based method, is evaluated after being retrained with additive Gaussian noise at a specific standard deviation to ensure a fair comparison. Moreover, a modified version of DGCNN [31] is used as an additional baseline, which replaces the segmentation head with a single-point convolution to regress the point displacement.

The proposed method performance is evaluated using the Chamfer measure, also known as Cloud-to-Cloud (C2C) distance, as a metric. This metric is commonly used in point cloud denoising as it calculates the average distance between denoised points and the original surface. The Chamfer measure is computed by finding the mean distance between each denoised point and its closest ground truth point, and the mean distance between each ground truth point and its closest denoised point, then taking their average:

$$\mathrm{C2C} = \frac{1}{2N}\left[\sum_{i=1}^{N} \min_{j} \|\hat{\mathbf{x}}_i - \mathbf{x}_j\|_2^2 + \sum_{j=1}^{N} \min_{i} \|\hat{\mathbf{x}}_i - \mathbf{x}_j\|_2^2\right]. \tag{3.3}$$

The results of the experiments at different noise levels are reported in Table 3.1, 3.2 and 3.3. As described in Sec. 3.2.3, in the proposed network we consider two different loss functions obtaining two versions of the proposed method, namely GPDNet MSE and GPDNet MSE-SP.

Table 3.1 GPDNet: Denoising results evaluated in terms of chamfer measure ($\times 10^{-6}$), for $\sigma = 0.01$, network with 16-NN.

Class	Noisy	DGCNN [31]	APSS [47]	RIMLS [46]	AWLOP [49]	MRPCA [53]	GLR [54]	PCN [57]	**GPDNet MSE**	**GPDNet MSE-SP**
airplane	50.32	44.82	28.22	39.73	31.27	28.19	19.56	26.36	**17.22**	17.58
bench	48.71	38.70	26.97	32.76	34.08	32.93	20.43	27.64	**19.33**	19.80
car	64.34	60.47	47.73	55.56	54.21	44.33	42.22	75.34	**38.09**	38.14
chair	60.78	59.69	37.31	45.65	47.91	38.41	34.98	55.10	**29.50**	29.69
lamp	59.73	52.54	24.57	34.02	35.23	31.51	19.67	20.58	**16.17**	17.15
pillow	69.79	64.28	**15.64**	21.23	46.36	23.95	17.59	21.07	17.11	19.04
rifle	38.97	26.99	36.01	49.37	27.79	23.49	15.84	15.09	14.45	**14.00**
sofa	69.63	65.05	**22.27**	28.04	53.08	32.14	30.88	43.36	25.87	27.21
speaker	73.50	68.72	**26.50**	30.19	58.92	47.57	40.78	76.09	34.87	35.81
table	56.21	50.17	27.45	32.63	41.26	34.78	27.12	43.02	**24.27**	24.64
avg.	59.20	53.14	29.27	36.92	43.01	33.73	26.91	40.36	**23.69**	24.31

Table 3.2 GPDNet: Denoising results evaluated in terms of chamfer measure ($\times 10^{-6}$), for $\sigma = 0.015$, network with 16-NN.

Class	Noisy	DGCNN [31]	APSS [47]	RIMLS [46]	AWLOP [49]	MRPCA [53]	GLR [54]	PCN [57]	**GPDNet MSE**	**GPDNet MSE-SP**
airplane	97.78	84.40	86.42	106.33	73.32	67.39	36.76	35.27	28.47	**27.62**
bench	94.82	64.76	75.51	91.93	82.04	70.05	32.19	30.10	28.72	**26.96**
car	102.23	93.43	72.56	103.52	93.38	69.88	55.92	92.23	52.92	**51.77**
chair	105.16	94.45	81.47	104.38	92.47	73.45	48.62	69.18	46.28	**43.73**
lamp	120.65	112.06	65.79	82.40	88.78	77.09	39.93	30.59	**27.37**	28.60
pillow	132.57	113.32	22.74	42.54	112.54	73.67	31.38	29.02	**23.32**	27.25
rifle	80.40	61.04	92.14	110.51	69.35	55.65	31.81	**21.45**	28.43	22.48
sofa	121.02	99.63	42.80	69.92	107.58	72.62	51.12	61.15	**40.10**	42.04
speaker	123.27	114.12	46.45	58.28	110.29	77.95	53.75	87.68	**49.20**	49.57
table	103.50	84.95	62.64	78.21	89.33	70.87	37.94	43.88	36.06	**33.89**
avg.	108.14	92.22	64.85	84.80	91.91	70.86	41.94	50.05	36.09	**35.39**

Table 3.3 GPDNet: Denoising results evaluated in terms of chamfer measure ($\times 10^{-6}$), for $\sigma = 0.02$, network with 16-NN.

Class	Noisy	DGCNN [31]	APSS [47]	RIMLS [46]	AWLOP [49]	MRPCA [53]	GLR [54]	PCN [57]	**GPDNet MSE**	**GPDNet MSE-SP**
airplane	161.79	127.44	175.68	186.24	145.94	123.71	90.55	74.17	45.96	**42.30**
bench	161.52	99.36	166.85	182.42	157.29	127.51	83.99	90.34	41.24	**36.77**
car	148.74	113.94	141.69	167.78	145.51	109.49	77.56	160.08	72.06	**67.43**
chair	163.75	132.91	160.01	155.38	158.12	122.70	79.85	145.56	67.91	**60.16**
lamp	204.05	153.02	178.08	198.22	187.31	146.41	109.24	85.31	45.21	**44.60**
pillow	215.58	190.32	164.83	196.53	206.14	150.65	85.86	92.84	**34.47**	38.58
rifle	144.18	131.91	195.68	176.07	144.22	105.87	89.19	71.57	43.07	**29.55**
sofa	184.11	155.51	166.34	190.91	178.93	133.98	89.31	144.72	**62.58**	65.06
speaker	186.01	136.72	138.80	162.34	180.45	126.17	84.37	160.26	66.57	**63.40**
table	168.32	115.00	171.25	179.81	162.36	125.72	78.06	102.17	50.47	**44.80**
avg.	173.80	135.61	165.92	179.57	166.63	127.22	86.80	112.70	52.95	**49.265**

The results clearly indicate that both versions of the proposed method outperform state-of-the-art methods, particularly at medium and high levels of noise, as demonstrated in Table 3.2 with$\sigma = 0.015$ and Table 3.3 with $\sigma = 0.02$. However, at low noise levels, other algorithms become more competitive, and the performance gap decreases, but the proposed method still achieves the best results in most categories, as reported in Table 3.1. This is due to the fact that most other methods involve surface reconstruction or normal estimation, which cannot be computed with sufficient accuracy at high levels of noise, while the proposed method directly estimates the denoised point cloud. Furthermore, it is observed that the GPDNet MSE-SP version is particularly effective at high levels of noise, outperforming GPDNet MSE in almost all categories, as shown in Table Table 3.3. This is due to the regularizing effect of the surface distance component of the loss, which is especially useful at high noise variance because it can incorporate more prior knowledge about the data. The performance difference between the two variants decreases at low noise levels, as shown in Table 3.1. It is important to note that DGCNN shows poor performance for denoising tasks, as it was originally designed for classification or segmentation. This is consistent with the results presented in the PointCleanNet paper [57] emphasizing the importance of the proposed method design for denoising tasks. In particular, there are several factors that make DGCNN unsuitable for point cloud denoising. Firstly, the spatial transformer block used in DGCNN is not appropriate for denoising because it seeks a global representation while denoising is focused on local representations of point neighborhoods. Moerover, it significantly increases the computational complexity for large point clouds. Secondly, the graph convolution operation in DGCNN uses a max operator for aggregation, which is not stable in the presence of noise. Thirdly, the specific graph convolution definition used in DGCNN is less general than the one presented in this research, which allows for adaptive filters and edge attention terms that are especially important for denoising in the presence of noise.

Furthermore, a different metric is used to evaluate the effectiveness of our denoising method. Specifically, we assess whether an existing algorithm for estimating surface normals can produce more accurate results when applied to point clouds denoised by our method. Since surface normals are important for many applications, evaluating their quality extracted from denoised data is a relevant way to assess a denoiser performance. A different test set is used for this experiment, consisting of five well-known point clouds with available ground truth normals: Armadillo, Bunny, Column, Galera and Tortuga. For each denoising method, we compute the unoriented normal vector for each point in the denoised point cloud and compare it to the ground truth normals using the built-in MATLAB function based on principal component analysis. The unoriented normal angle error (UNAE) is computed as

$$\mathrm{UNAE} = \frac{1}{N}\sum_{i=1}^{N} \arccos\left[1 - \frac{1}{2}\min\left(\|\hat{\mathbf{n}}_{i^*} - \mathbf{n}_i\|_2^2, \|\hat{\mathbf{n}}_{i^*} + \mathbf{n}_i\|_2^2\right)\right], \tag{3.4}$$

where $\mathbf{n}_i$ is the groud-truth normal vector at $\mathbf{x}_i$ and $\hat{\mathbf{n}}_{i^*}$ is the estimated normal vector at the denoised point closest to $\mathbf{x}_i$. Table 3.4 shows the average error over the five test point clouds. Note that the minimum error is around six degrees, which is attributed to the estimation error introduced by the MATLAB algorithm used for normal estimation, as shown in the first column of Table 3.4. The proposed denoising method, particularly the variant that only employs Mean Squared Error (MSE) as the loss function, improves the accuracy of normal estimation and performs better than the state-of-the-art at each noise level examined. Moreover, it is interesting to note that machine learning-based methods are more resilient to noise than model-based methods, as their performance deteriorates more gradually as the noise level increases.

Table 3.4 GPDNet: Denoising results evaluated in terms of Unoriented normal angle error (degrees), network with 16-NN.

σ	Clean	Noisy	DGCNN [31]	APSS [47]	RIMLS [46]	AWLOP [49]	MRPCA [53]	GLR [54]	PCN [57]	**GPDNet MSE**	**GPDNet MSE-SP**
0.01	6.44	31.13	30.83	22.60	24.52	29.79	31.40	21.90	26.85	**20.11**	22.33
0.015	6.44	32.77	32.52	31.83	37.35	32.17	39.97	25.99	27.54	**21.16**	24.46
0.02	6.44	33.77	32.31	42.42	45.86	33.41	42.45	31.30	28.65	**22.78**	27.06

Fig. 3.2 displays the denoised point cloud for each method at a medium level of noise, along with the surface distance of each point for a better understanding of the denoised points position relative to the ground truth. The root mean square value of the surface distance (RMSD) can be calculated as:

$$\mathrm{RMSD} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \min_{j} \|\hat{\mathbf{x}}_i - \mathbf{x}_j\|_2^2}. \tag{3.5}$$

On average, both versions of the proposed method show a lower points-surface distance, indicating that the reconstructed point cloud is closer to the original one. Another qualitative comparison is displayed in Fig. 3.3, which shows the unoriented normal estimation error for each denoised point. The proposed method, particularly the version with only MSE, demonstrates lower normal estimation errors, suggesting that the denoised point cloud is of higher quality.

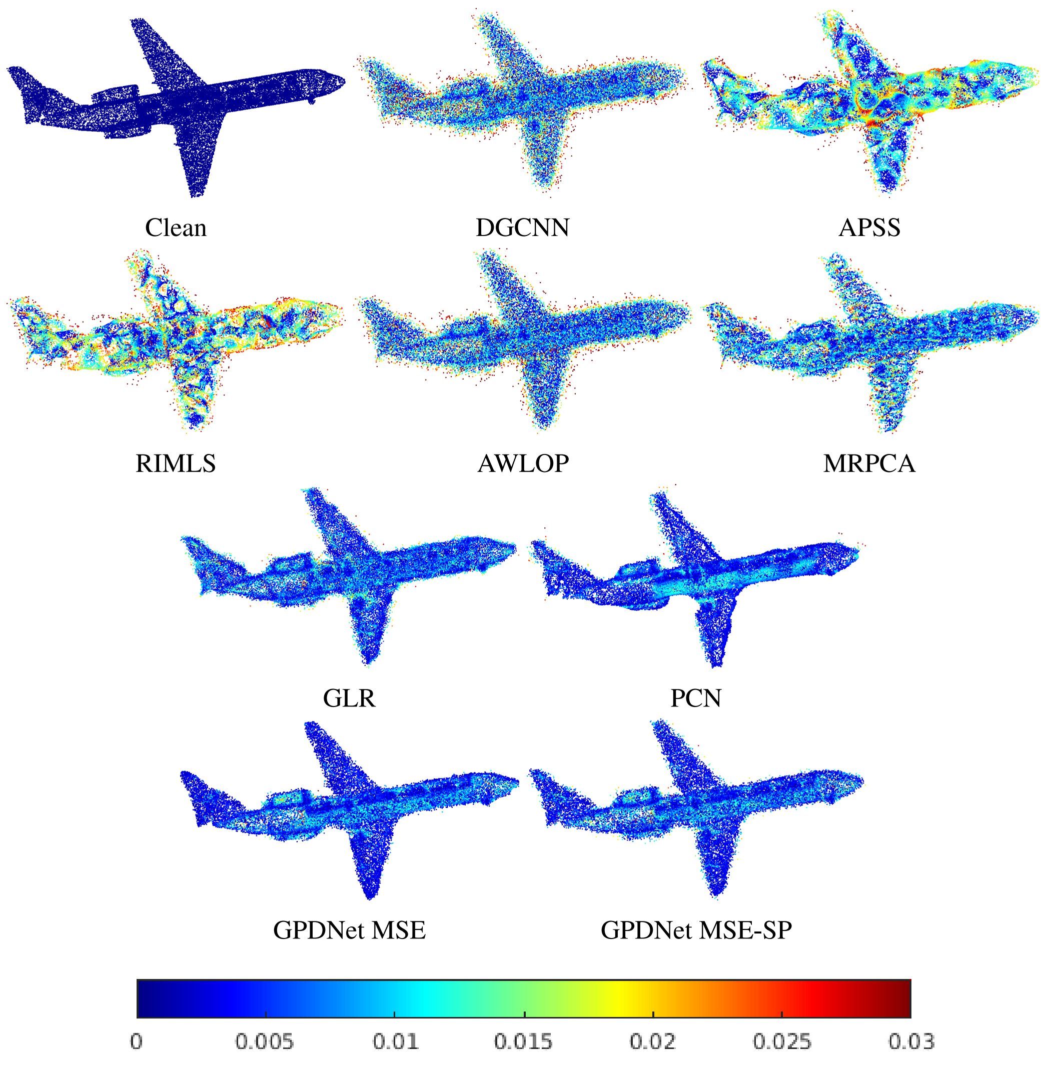


Fig. 3.2 GPDNet: Denosing results for $\sigma = 0.015$, network with 16-NN. Color represents distance to surface (red is high, blue is low). Top left to bottom right: clean point cloud, DGCNN (RMSD = 0.0091), APSS (0.0123), RIMLS (0.0127), AWLOP (0.0106), MRPCA (0.0096), GLR (0.0070), PointCleanNet (0.0065), GPDNet MSE (0.0060), GPDNet MSE-SP (0.0062).

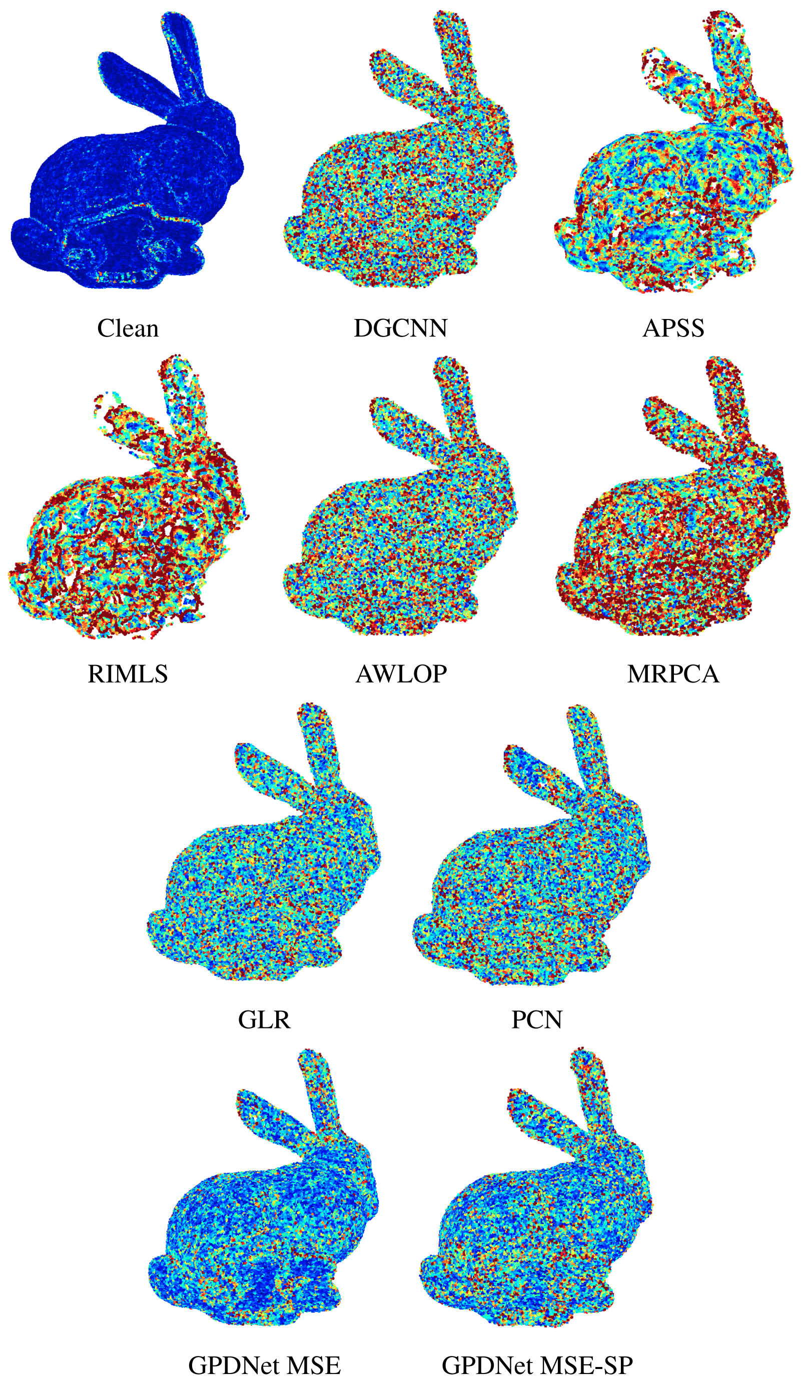


Fig. 3.3 GPDNet: Denosing results for $\sigma = 0.015$, network with 16-NN. Color represents unoriented normal angle error (red is high, blue is low). Top left to bottom right: clean point cloud (UNAE = $3.75°$), DGCNN ($29.73°$), APSS ($26.29°$), RIMLS ($33.63°$), AWLOP ($29.18°$), MRPCA ($37.50°$), GLR ($22.08°$), PointCleanNet ($23.63°$), GPDNet MSE ($16.62°$), GPDNet MSE-SP ($23.11°$).

3.3.3 Ablation studies

It is examined how GPDNet performs based on some design choices. Initially, we analyze the effect of dynamic graph computation, which involves updating the graph from the hidden feature space, as shown in Fig. 3.1, as opposed to constructing a fixed graph in which neighbors are determined in the noisy 3-D space and used for all graph-convolutional layers. Table 3.5 reveals that dynamic graph update enhances performance due to improved neighbor selection.

Table 3.5 GPDNet: Fixed vs. Dynamic graph, for $\sigma = 0.015$, network with 8-NN.

	GPDNet MSE		**GPDNet MSE-SP**	
	Dynamic	Fixed	Dynamic	Fixed
C2C ($\times 10^{-6}$)	**35.68**	37.00	**36.99**	38.45
UNAE (degrees)	**23.56**	23.75	**26.29**	26.65

The study also investigated the effect of neighborhood size on the denoising performance. Increasing the number of neighbors in the graph convolution operation can capture more context and potentially denoise smooth areas in the point cloud, but at the cost of reduced localization and increased computational complexity. This phenomenon is similar to the results in image denoising where the optimal receptive field size depends on the noise variance. Results in Tables 3.6 and 3.7 demonstrate that increasing the number of neighbors is advantageous up to a point of saturation. Furthermore, it was observed that the impact of a larger receptive field is more significant for the GPDNet MSE-SP variant.

Table 3.6 GPDNet: Number of neighbors (Chamfer measure $\times 10^{-6}$).

		4-NN	8-NN	16-NN	24-NN
$\sigma = 0.01$	GPDNet MSE	28.27	24.43	**23.69**	23.84
	GPDNet MSE-SP	30.38	25.54	24.31	24.44
$\sigma = 0.015$	GPDNet MSE	40.46	35.68	36.09	36.67
	GPDNet MSE-SP	46.05	36.99	**35.39**	35.80
$\sigma = 0.02$	GPDNet MSE	58.88	50.34	52.96	55.45
	GPDNet MSE-SP	64.63	51.82	**49.26**	50.43

Table 3.7 GPDNet: Number of neighbors (UNAE - degrees).

		4-NN	8-NN	16-NN	24-NN
$\sigma = 0.01$	GPDNet MSE	27.22	22.51	**20.11**	20.89
	GPDNet MSE-SP	29.04	24.10	22.33	22.16
$\sigma = 0.015$	GPDNet MSE	28.03	23.56	**21.16**	21.18
	GPDNet MSE-SP	31.31	26.29	24.46	23.80
$\sigma = 0.02$	GPDNet MSE	31.09	25.67	**22.78**	22.98
	GPDNet MSE-SP	32.00	28.81	27.06	26.92

3.3.4 Feature analysis

In this section, the characteristics of the receptive field are examined, which refers to the set of points that influence the features of a specific point due to the graph convolutional layers. In Fig. 3.4 an example of the receptive field of a single point for the output of the graph convolutional layers of a residual block is showed. It can be observed that the receptive field is localized in the 3-D space, and its size increases as the number of layers increases. Moreover, since the graph is dynamically constructed in the feature space, the points of the receptive field are not just the spatially closest ones but also among the ones with similar shape characteristics. For example, in Fig. 3.4 the considered point is on the lower side of the chair stretcher and all the points of the receptive field belong to the same part of the surface.

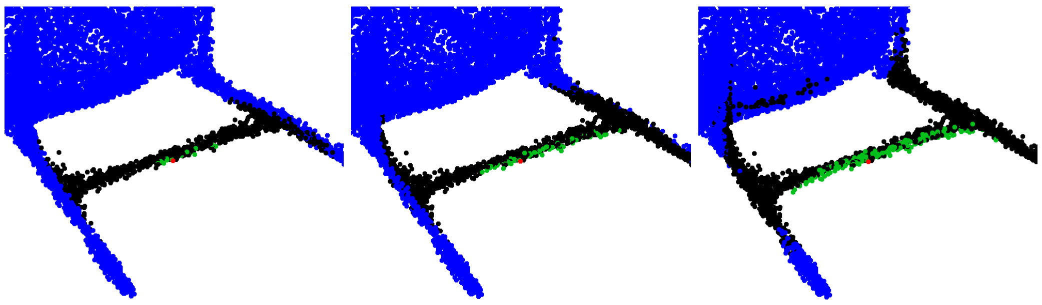


Fig. 3.4 GPDNet: Receptive field (green) and search area (black) of a point (red) for the output of the three graph-convolutional layers of the second residual block of the network with respect to the input of the first graph-convolutional layer in the block. Effective receptive field size: 16, 65, 189 points.

To better analyze this non-local property of the receptive field the radius of the receptive field in the 3-D space is measured and compared to a fixed graph construction where the neighbors are determined by proximity in the noisy 3-D space. Fig. 3.5 shows the radius of the receptive field of each point at the output of a residual block with respect to the input of the residual block. he radius is evaluated as the 90 percentile Euclidean distance in the 3D space on the clean point cloud (90 percentile is used since the maximum might be an unstable metric). They found that when using dynamic graph construction, the radius is only slightly larger in the first residual block but can be significantly larger in the second one. This implies that the feature space is building and exploiting more non-local features with patterns similar to those observed in Fig. 3.4.

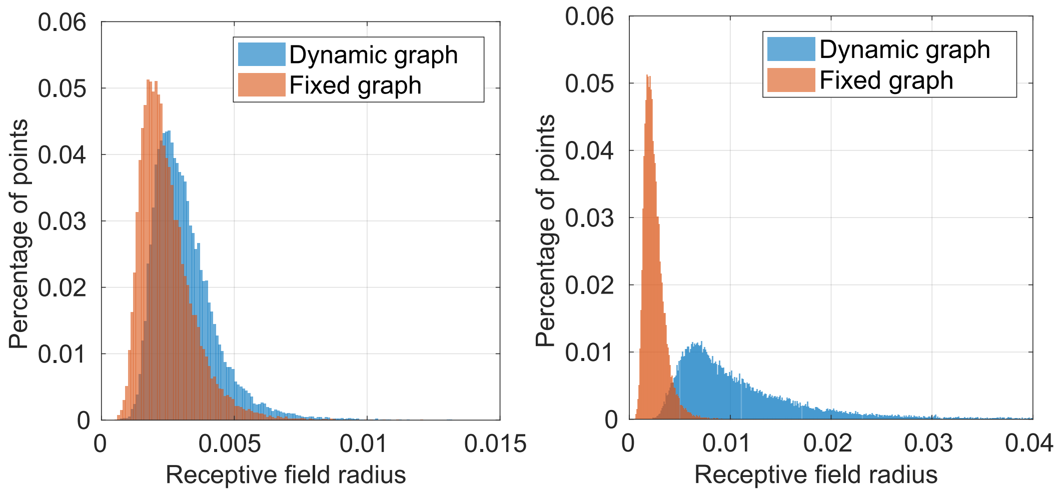


Fig. 3.5 GPDNet: Radius of receptive field of points at the output of residual block with respect to its input. Left: first residual block. Right: second residual block. Neighbor selection in the noisy 3D space for fixed graph and in the feature space for dynamic graph. Radius is measured as the 90 percentile Euclidean distance to the points in the receptive field on the clean 3D point cloud.

3.3.5 Structured noise

To test if the proposed architecture can work on other types of noise, the network has been trained on a simulated LiDAR dataset using the Blensor software [84].. The simulated scanning of the Shapenet objects with a Velodyne HDL-64E scanner includes two types of noise: a laser distance bias with Gaussian distribution and a per-ray Gaussian noise. Both distributions have a zero-mean and a standard deviation of 1% of the longest side of the object bounding box. Also PointCleanNet has been trained on the simulated data for comparison with the proposed method. The results, shown in Table 3.8, are consistent with those obtained with white Gaussian noise, with the proposed method outperforming PointCleanNet. RMSD is used instead of the Chamfer measure as the metric since the points are not uniformly distributed.

Table 3.8 GPDNet: Velodyne scan structured noise evaluated in term of RMSD, network with 8-NN.

Noisy	PointCleanNet	**GPDNet MSE**	**GPDNet MSE-SP**
0.1447	0.0966	0.0664	**0.0602**

3.4 Conclusions

This work introduced a new type of neural network for denoising point clouds. The network utilized graph-convolutional layers and was able to learn complex features in a fully convolutional manner. The results of the experiments showed that the proposed method outperformed existing state-of-the-art techniques and demonstrated robustness against high levels of noise and structured noise distributions commonly found in real-world LiDAR scans.

Chapter 4

GPOD: Graph convolutional neural networks for robust point cloud outlier removal and denosing

4.1 Introduction

In this research the method described in Chapter 3 is significantly expanded and graph convolutional neural network are further investigated for restoration tasks. First, it is considered the problem of simultaneous denoising and outlier detection and removal instead of simple denoising of white Gaussian noise. A novel neural network architecture able to efficiently create a shared feature space both for the outlier detection and denoising tasks is proposed. Notably, in contrast to [10], the denoising method is blind, and therefore it does not require variance-specific training, creating a generic framework that can be applied to a large variety of challenging restoration scenarios.

The main contributions of this research when compared to previous works are:

- A new architecture for improving the geometry of point cloud data that is able to identify and remove outliers as well as denoise the remaining data points in a single model.
- An architecture that is able to efficiently perform the denoise task without any prior knowledge of the amount of noise.

- A graph convolutional layer that is robust to noise and able to adapt to the data by dynamically updating the graph.

- State-of-the-art performance in denoising point cloud data.

4.2 Proposed method

This section presents the proposed Graph-convolutional Point Outlier removal and Denoising (GPOD) network. The goal is to create a model that can effectively remove noise from point clouds that are impacted by both outlier and geometry noise. First a brief overview of the system is given and then a more in-depth description of the primary components is reported.

4.2.1 Architecture overview

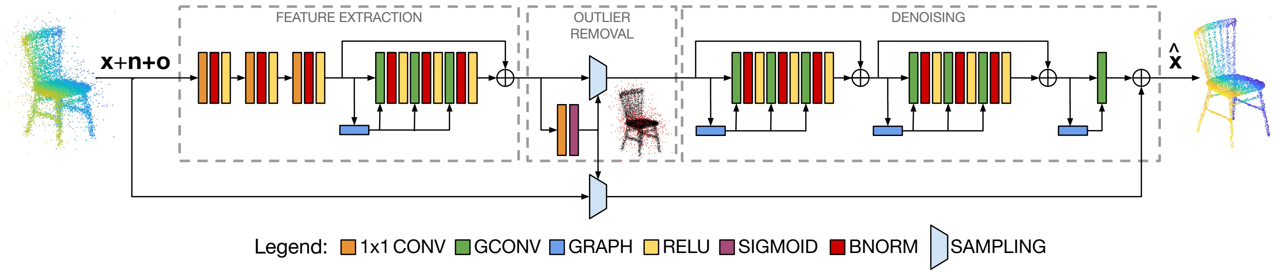

FEATURE EXTRACTION
OUTLIER REMOVAL
DENOISING
x+n+o
$\hat{\mathbf{x}}$
Legend: 🟧 1x1 CONV 🟩 GCONV 🟦 GRAPH 🟨 RELU 🟪 SIGMOID 🟥 BNORM SAMPLING

Fig. 4.1 Graph-convolutional Point Outlier removal and Denoising (GPOD) architecture. The network takes as input the noisy point cloud with outliers $\mathbf{x}+\mathbf{n}+\mathbf{o}$ and provides as output the denoised point cloud $\hat{\mathbf{x}}$. The network first projects the noisy point cloud into a feature space by means of 1-by-1 convolutions (1x1 CONV) and extract useful geometric features with several graph convolutional layers (GCONV) in the feature extraction step. Then the learned features are fed into the outlier removal module and the outliers are detected by means of a simple binary classifier made of 1-by-1 convolution. Then, the features are sampled to remove points that are classified as outliers (SAMPLING) and the remaining points are passed to the denoising module. This module estimates the additive noise for each point that is projected back into the 3-D space by a single graph-convolutional layer. The estimated noise is finally subtracted to the sampled noisy input (the points classified as outliers are excluded) and the desired denoised point cloud is obtained.

The proposed method, shown in Fig. 4.1, is a single model able to simultaneously tackle the task of outlier removal and denoising, sharing a common feature space.

The network exploits as core building blocks a graph-convolutional layer that implement the Lightweight ECC [34], described in Sec. 2.2.2. The graph-convolutional layer is designed as the one proposed and described in Sec. 3.2.2. At high-level the network can be divided into three sections: the shared feature extraction, the outlier detection and removal and finally the denoising.

The first step of the method is to extract features from the point cloud data that captures the local characteristics of the data. These features are used for both identifying and removing outliers points and denoising the remaining data points and are robust to noise in the input data. This stage consists of three layers of convolutional operations using single-point convolutions and a block of three graph-convolutional layers with residual connections.

Then, the process of identifying data points outside if the original shape, the outliers, is implemented by using the features extracted from the point cloud, and feeding them into a point-by-point binary classifier, which is simply designed as a single-point convolution. Then, the point cloud is down-sampled by removing all the points that are classified as outliers, based on a pre-determined threshold for the classifier's probability output. Note that the threshold is not an information related the amount of noise inserted in the data that has to be known, but it is simply chosen to maximize the F1 score on the validation set where different levels of outlier noise are considered. Therefore, since the threshold is automatically determined it is not in contrast with the blind training.

After the outlier removal process, the remaining points are processed by the rest of the network, which is responsible for denoising task. This is done by using two residual blocks of graph-convolutional layers with residual connections.

One of the strength of the method is that it is able to perform outlier removal and denoising with a single model, differently to other methods available in the literature, such PointCleanNet [57]. Furthermore, this design choice as several advantages. Sharing a common parameterization for both tasks increases efficiency, which is a benefit. Moreover, removing outliers early is advantageous because it helps produce better final results by returning outlier probabilities and denoised data. Outliers can have a negative impact on the network's feature space and reduce performance. Outlier points don't contain any useful information, and if not removed, they can interfere with graph-convolution operations and affect displacement estimation, especially when there's a high level of noise. Therefore, it's important to disregard outlier points to avoid these issues.

4.2.2 Outlier detection

This part of the network is composed of two building blocks: the feature extraction and the outlier removal block, as reported in Fig. 4.1. The feature extraction process is performed by using three convolutional operations that gradually project the input data from 3-D space to a feature space of F dimensions, and a residual block. A residual block is a fundamental component of the proposed network and is composed of three graph-convolutional layers followed by normalization to improve the stability of the training process, and a skip-connection between the input and output to reduce the risk of vanishing gradients. The graph is shared among the graph-convolutional layers inside the residual block to limit computational complexity. Then, the learned features are processed by a binary classifier, which is implemented as a single-point convolution that returns the probability of each point being an outlier. All the points that are considered as outliers according to a specific detection threshold are removed from the feature representation of the original point cloud. The feature vectors of the remaining points are then fed as the input for the denoising block. The threshold is chosen to maximize the F1 score on the validation set. The loss function considered for the outlier detection task is the weighted cross entropy (WCE). This function is a measure of the dissimilarity between the predicted probability distribution and the true distribution. It is commonly used in supervised machine learning tasks, such as classification, where the goal is to minimize the dissimilarity between the predicted class probabilities and the true class labels. The weighted cross-entropy loss function is a variant of the standard cross-entropy loss function that assigns different weights to each sample or class, depending on their importance or rarity in the dataset. This is done by multiplying the cross-entropy loss for each sample by a weight factor. This weight factor can be used to balance the loss for each class and to give more importance to some class or to under-represented samples. In the proposed scenario the number of outliers is typically far less compared to the total number of points in the point cloud, therefore the weighted cross entropy is the most reasonable choice to supervise the task for outlier removal:

$$L_{\text{WCE}} = -\sum_i \left[\beta \cdot p_i \log(\hat{p}_i) + (1 - p_i) \log(1 - \hat{p}_i)\right], \tag{4.1}$$

where β is a positive weight, p_i is the true label of the i-th point and $\hat{p}_i$ is the predicted probability of being an outlier. The parameter β is a weight able to penalize or increment the cost of positive error with respect to negative errors. If $\beta > 1$, the number of false negatives decreases increasing the recall, otherwise the number of false positive decreases increasing the precision. In the proposed network the parameter β is set to a value larger than one in order to increase the chance of capturing the outliers, avoiding false negatives that would highly penalize the overall denoising performance.

4.2.3 Denoising with outlier removal

The denoising block reported in Fig. 4.1 takes as input the feature representations of all the points except those classified as outliers, and returns as output the final denoised point cloud in 3-D space without outliers.

The overall structure of the block architecture is based on a residual network. Instead of estimating the denoised point cloud, the network estimates the additive noise component of the input because previous research has demonstrated [82] that predicting the residual is easier than directly denoising the data. The noise estimation is performed in the feature space using two residual blocks, and the estimated noise is projected to the 3-D space by a single graph-convolutional layer. Finally, the estimated noise is removed from the noisy point cloud. A more detailed explanation of this block can be found in Sec. 3.2. At the beginning of each residual block, the graph is created by choosing k nearest neighbors for each point based on the Euclidean distances in the feature space. The dynamic graph construction, where the graph is updated after each residual block, has been demonstrated [31, 64] to generate more effective feature representations. Furthermore, it has been shown to be particularly beneficial in the context of a residual denoising network since it helps to uncover latent correlations that may not have been removed yet.

The loss function used for denoising combines two terms: mean squared error (MSE) and surface proximity (SP) regularization. The surface proximity term measures the distance between each denoised point and the closest ground truth point. The loss function is then defined as follow:

$$L_{\mathrm{MSE-SP}} = \frac{1}{N}\sum_{i=1}^{N}\left[||\hat{\mathbf{x}}_i^{'} - \mathbf{x}_i||_2^2 + \lambda \min_j ||\hat{\mathbf{x}}_i^{'} - \mathbf{x}_j||_2^2 \right], \tag{4.2}$$

where N is the number of points in the original point cloud, $\hat{\mathbf{x}}^{'}$ is the denoised point cloud without the outliers, $\mathbf{x}$ is the original point cloud without additive noise and outliers, and λ is a regularization hyperparameter. The possible remaining outliers, which are not detected by the outlier classifier, are not considered in the loss function since its purpose is to constrain the denoising performance in presence of additive Gaussian noise. The loss function reported in Eq. 4.2 is designed just to constrain the denoising task, and the outliers points, whether they are correctly recognized or not, are excluded. This design choice is made in order to let the network to solely focus and learn to denoise points and not be disturbed by points that are outliers not correctly classified that constitute much more challenging points to denoise. Moreover, the proposed method is in a supervised setting and the ground truth original positions for outliers are not available since they are artifacts.

4.2.4 Training procedure

The final loss for denoising with outlier removal combines the losses for the two previously described tasks as:

$$L_{\mathrm{OR-DN}} = L_{\mathrm{MSE-SP}} + \alpha L_{\mathrm{WCE}}, \tag{4.3}$$

where α is a regularization parameter.

This work's training process involves three steps:

- Initially, only the feature extraction and outlier removal block's parameters are optimized using Eq. (4.1) as the loss function.
- Then, the feature extractor and the outlier removal blocks are freezed and just the parameters of the denoising block are optimized using Eq. (4.2) as the loss function. In such way the denoising branch can specialize on his specific task.

- Finally, all the parameters of the proposed network are fine-tuned using Eq. (4.3) as the loss function. In this phase the network is trained end-to-end and all the branches learn to work together, especially the feature extractor can learn a feature space useful for both the outlier detection and denoising task.

This approach allows each module to specialize for its own tasks while reducing overall computational complexity. Additionally, it provides the benefits of end-to-end optimization in the final fine-tuning step.

4.3 Experimental results

4.3.1 Experimental setting

The training, validation and test sets are collections of post-processed point clouds from the Shapenet [2] repository. This database is composed of 3D models of 55 object categories, each described as a collection of meshes. The dataset creation pipeline is similar to the one implemented for GPDNet, described in Chapter 3. The point clouds are obtained by sampling 30720 points from each selected models, which are then scaled and normalized. The training set consists of over 100000 patches of 1024 points each, randomly selected from point clouds across different categories, except for those reserved for testing and outlier validation sets. Each patch is created by selecting a point and its 1023 closest points. The test set contains 100 point clouds from ten different categories:airplane, bench, car, chair, lamp, pillow, rifle, sofa, speaker, and table. It is divided into two subsets: one with only Gaussian noise and the other with additional outliers. The validation set consists of 10 point clouds from five different categories: bath, clock, laptop, tower and train. It is used to set the threshold for outlier detection. The threshold is determined by maximizing the F1 score over the validation set and is used in the subsequent training stages. For the following experiments the threshold is set to 0.03.

The obtained sets used for training, validation and testing are artificially corrupted with noise to simulate real-world scenarios. To do this, Gaussian noise is added to the original data with different levels of standard deviation ranging from 0.01 to 0.02. The proposed method is designed to handle blind denoising, meaning that it does not require knowledge of the standard deviation of the noise. Therefore, data corrupted by all standard deviations in this range are used for training. In the denoising with outlier removal task, outliers are modeled as points corrupted by Gaussian noise with a standard deviation of 0.2, which is 10 times higher than the highest level of Gaussian noise added to the data. These outliers are added to 10% of the noiseless data points. If an outlier is is closer than one noise standard deviation to the original position, it is resampled.

The suggested training process consists of three stages, with each stage trained for around 50000, 80000, and 100000 iterations, respectively, using a batch size of 16 patches. Most layers use 99 features, except for the first three single-point convolutional layers of the feature extraction block, which gradually increase from 33 to 66, then finally to 99. The Adam optimizer is used, with a fixed learning rate of 10^{-5} for the first two stages and 10^{-6} for the last one. In the graph-convolutional implementation, a rank of 11 is set for low-rank approximation, and a circulant matrix is constructed with three circulant rows. During testing, the entire point cloud is fed as input to the network, and each point is associated with a search area for identifying neighbors. By default, 16 nearest neighbors are used for graph construction.

4.3.2 Outlier detection performance

This section evaluates the GPOD networks ability to detect outliers and compares it to other methods. The comparison includes a statistical method (STM) based on [85], as well as two learning-based methods: PointCleanNet [57] and DGCNN [31]. PointCleanNet is a recent method that addresses both denoising and outlier removal tasks using two separate networks. However, in this section, only the performance of the outlier detection network is considered. DGCNN is a point cloud segmentation and classification method that has been modified for outlier detection. It's worth noting that both PointCleanNet and DGCNN only perform outlier detection, while GPOD is a single model that performs both outlier detection and denoising.

The performance of different outlier detection techniques is evaluated using Receiver Operating Characteristic (ROC) and precision/recall curves, which are presented in Fig. 4.2 and 4.3, respectively. Moreover, the F1-score, recall, and precision of the ten categories in the test set are reported in Table 4.1. To ensure a fair comparison, the detection threshold is chosen as the one that maximizes the F1 score over the validation set for all the methods. For instance, the threshold is 0.8 for PointCleanNet, 4.4×10^{-6} for DGCNN, and 0.61 for STM.

The findings indicate that all techniques demonstrate similar performance, which is noteworthy because GPOD is not solely designed for outlier removal. It is important to note that PCN has a tendency to have a high recall at the expense of lower precision, which is a consequence of its design.

Table 4.1 GPOD: Outlier detection performance (%).

		F1		
σ	STM [85]	DGCNN [31]	PCN [57]	**GPOD**
0.02	89.58 ± 3.07	**89.69 ± 3.57**	81.01 ± 1.36	89.43 ± 3.29
0.015	**92.21 ± 2.51**	92.12 ± 2.80	85.67 ± 1.65	91.41 ± 2.68
0.01	**94.10 ± 2.07**	93.81 ± 2.26	90.99 ± 2.08	92.50 ± 2.34
		Recall		
σ	STM [85]	DGCNN [31]	PCN [57]	**GPOD**
0.02	84.61 ± 4.76	85.36 ± 4.33	**88.27 ± 4.75**	83.88 ± 4.48
0.015	86.67 ± 4.27	87.09 ± 4.08	**90.05 ± 4.20**	85.29 ± 4.23
0.01	88.54 ± 3.98	88.76 ± 3.85	**91.53 ± 3.77**	86.30 ± 4.02
		Precision		
σ	STM [85]	DGCNN [31]	PCN [57]	**GPOD**
0.02	94.55 ± 1.45	94.54 ± 2.97	75.06 ± 2.17	**95.87 ± 2.31**
0.015	97.73 ± 0.82	97.83 ± 1.60	81.85 ± 2.15	**98.59 ± 0.99**
0.01	99.66 ± 0.18	99.56 ± 0.36	90.57 ± 2.12	**99.76 ± 0.22**

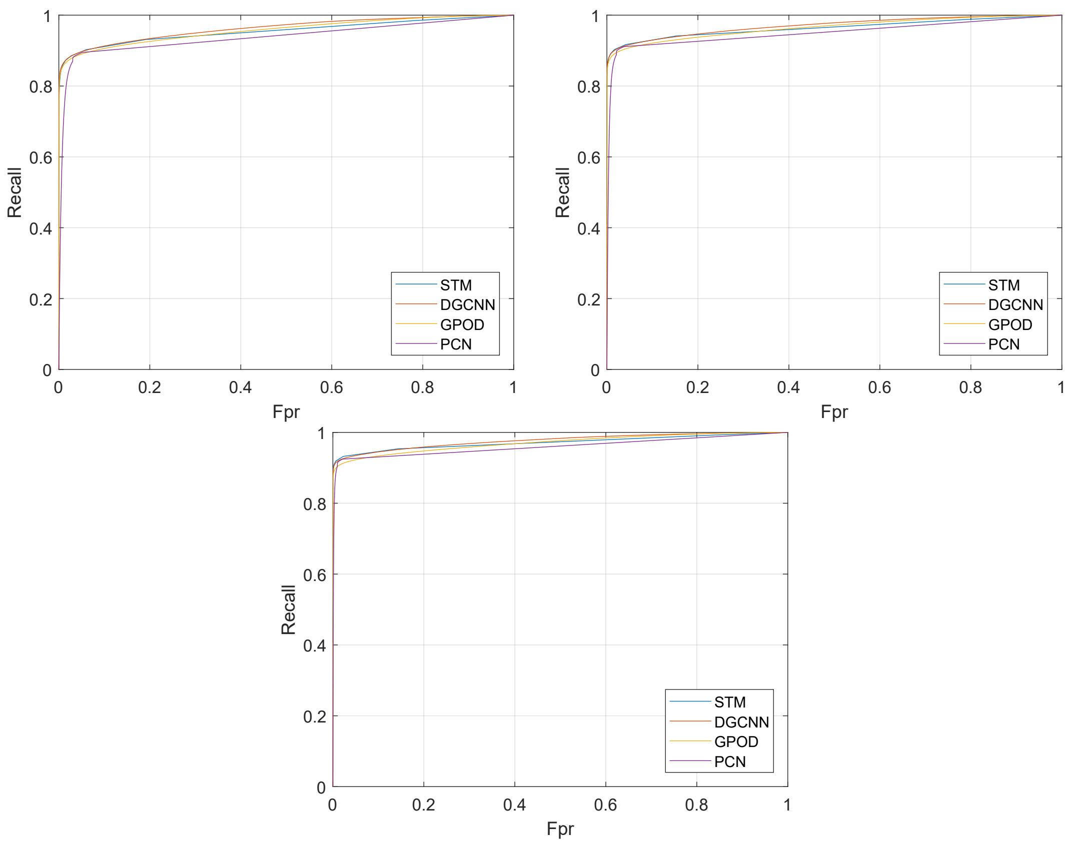


Fig. 4.2 GPOD: ROC curves. Left $\sigma = 0.02$, center $\sigma = 0.015$, right $\sigma = 0.01$.

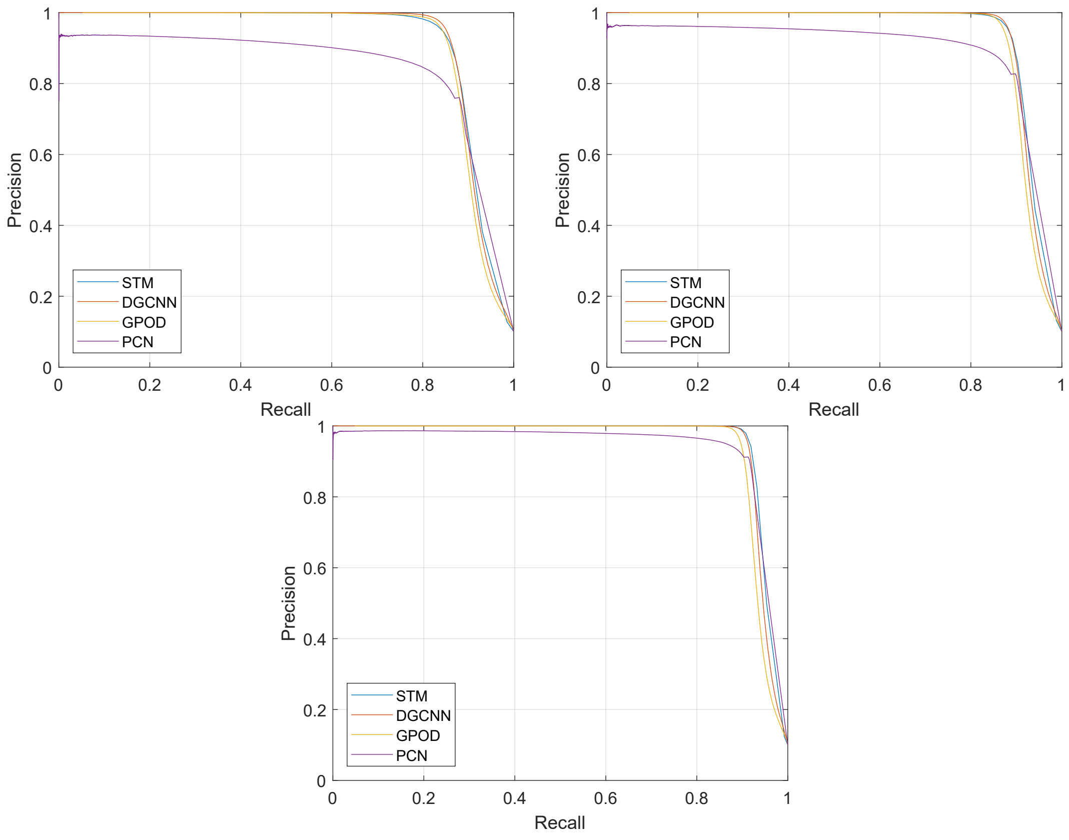


Fig. 4.3 GPOD: Precision-Recall curves. Left $\sigma = 0.02$, center $\sigma = 0.015$, right $\sigma = 0.01$.

4.3.3 Denoising performance without outliers

In this section, the focus is on evaluating the denoising performance of various methods when the test point clouds contain only additive Gaussian noise without outliers.

Different classes of point cloud denoising methods, as discussed in Section 2.5.2, are considered, including MLS-based surface fitting methods like APSS [47] and RIMLS [46], the edge-aware surface fitting method AWLOP [49], the sparsity-based method MRPCA [53], and the graph signal processing method GLR[54]. Furthremore, the state-of-the-art learning-based method PointCleanNet (PCN) [57] and a modified version of the DGCNN [31] method are included as baselines. The performance is compared using the chamfer distance or the Cloud-to-Cloud (C2C) distance, which computes the average distance of the denoised points from the original surface. The chamfer distance is computed as the average distance between each denoised point and its closest ground truth point, and the average distance between each ground truth point and its closest denoised point:

$$\mathrm{C2C}=\frac{1}{2}\left[\frac{1}{N_1}\sum_{j=1}^{N_1}\min_{i}||\hat{\mathbf{x}}_i-\mathbf{x}_j||_2^2+\frac{1}{N_2}\sum_{i=1}^{N_2}\min_{j}||\hat{\mathbf{x}}_i-\mathbf{x}_j||_2^2\right], \tag{4.4}$$

where N_1 and N_2 are respectively the number of points in the original and in the denoised point cloud, $\hat{\mathbf{x}}$ the denoised points and $\mathbf{x}$ the original points. The average C2C is computed over all the models of each category in the test set and the results of the experiments at different noise levels are reported in Table 4.2, 4.3 and 4.4.

The results indicate that GPOD performs better than state-of-the-art methods for denoising point clouds with medium to high levels of noise, as presented in Tables 4.2 and 4.3 with noise variances of $\sigma = 0.02$ and $\sigma = 0.015$, respectively. On the other hand, traditional model-based methods are more competitive for denoising at low noise variances ($\sigma = 0.01$). The reason of this behaviour is that most of the other methods involves surface reconstruction or normal estimation, which are not accurate at high levels of noise. In contrast, GPOD and other learning-based methods can use more complex features that are more robust at high noise levels. It is important to note that GPOD was trained blindly, without any information about the noise levels, which explains why its performance is better at higher noise levels and average at lower levels. Therefore, it is reasonable to consider the method highly competitive, since it achieves high performance at higher level of noise, which is the most critical scenario, and average results at lower level of noise. However, if more information about the noise variance is available, non-blind models can be trained to improve performance, particularly at low noise levels. In Table 4.5, GPOD and other learning-based methods, DGCNN and PCN, are trained using point clouds corrupted by Gaussian noise with a specific standard deviation ($\sigma = 0.01$), and the proposed method clearly outperforms the others.

Table 4.2 GPOD: Denoising performance evaluated in terms of chamfer distance ($\times 10^{-6}$) for $\sigma = 0.02$.

Class	Noisy	DGCNN [31]	APSS [47]	RIMLS [46]	AWLOP [49]	MRPCA [53]	GLR [54]	PCN [57]	**GPOD**
Airplane	161.79	131.11	175.68	186.24	145.94	123.71	90.55	74.17	**53.65**
Bench	161.52	122.55	166.85	182.42	157.29	127.51	83.99	90.34	**48.26**
Car	148.74	137.25	141.69	167.78	145.51	109.49	77.56	160.08	**74.50**
Chair	163.75	159.69	160.01	155.38	158.12	122.70	79.85	145.56	**66.52**
Lamp	204.05	273.24	178.08	198.22	187.31	146.41	109.24	85.31	**63.72**
Pillow	215.58	198.95	164.83	196.53	206.14	150.65	85.86	92.84	**64.84**
Rifle	144.18	86.67	195.68	176.07	144.22	105.87	89.19	71.57	**35.60**
Sofa	184.11	155.88	166.34	190.91	178.93	133.98	89.31	144.72	**78.79**
Speaker	186.01	172.84	138.80	162.34	180.45	126.17	84.37	160.26	**74.00**
Table	168.32	144.88	171.25	179.81	162.36	125.72	78.06	102.17	**53.47**
Average	173.80	158.31	165.92	179.57	166.63	127.22	86.80	112.70	**61.33**

Table 4.3 GPOD: Denoising performance evaluated in terms of chamfer distance ($\times 10^{-6}$), for $\sigma = 0.015$.

Class	Noisy	DGCNN [31]	APSS [47]	RIMLS [46]	AWLOP [49]	MRPCA [53]	GLR [54]	PCN [57]	**GPOD**
Airplane	97.78	77.18	86.42	106.33	73.32	67.39	36.76	35.27	**29.14**
Bench	94.82	70.17	75.51	91.93	82.04	70.05	32.19	30.10	**27.62**
Car	102.23	94.82	72.56	103.52	93.38	69.88	55.92	92.23	**54.53**
Chair	105.16	100.93	81.47	104.38	92.47	73.45	48.62	69.18	**47.07**
Lamp	120.65	173.83	65.79	82.40	88.78	77.09	39.93	30.59	**28.57**
Pillow	132.57	120.43	22.74	42.54	112.54	73.67	31.38	29.02	**27.27**
Rifle	80.40	45.26	92.14	110.51	69.35	55.65	31.81	**21.45**	23.30
Sofa	121.02	101.30	42.80	69.92	107.58	72.62	51.12	61.15	**43.91**
Speaker	123.27	114.86	46.45	58.28	110.29	77.95	53.75	87.68	**51.12**
Table	103.50	87.84	62.64	78.21	89.33	70.87	37.94	43.88	**35.30**
Average	108.14	98.66	64.85	84.80	91.91	70.86	41.94	50.05	**36.78**

Table 4.4 GPOD: Denoising performance evaluated in terms of chamfer distance ($\times 10^{-6}$) for $\sigma = 0.01$.

Class	Noisy	DGCNN [31]	APSS [47]	RIMLS [46]	AWLOP [49]	MRPCA [53]	GLR [54]	PCN [57]	**GPOD**
Airplane	50.32	39.80	28.22	39.73	31.27	28.19	**19.56**	26.36	31.64
Bench	48.71	36.88	26.97	32.76	34.08	32.93	**20.43**	27.64	33.03
Car	64.34	60.77	47.73	55.56	54.21	44.33	**42.22**	75.34	54.06
Chair	60.78	58.00	37.31	45.65	47.91	38.41	**34.98**	55.10	50.75
Lamp	59.73	112.13	24.57	34.02	35.23	31.51	**19.67**	20.58	31.79
Pillow	69.79	62.97	**15.64**	21.23	46.36	23.95	17.59	21.07	29.21
Rifle	38.97	22.20	36.01	49.37	27.79	23.49	**15.84**	15.09	36.38
Sofa	69.63	57.87	**22.27**	28.04	53.08	32.14	30.88	43.36	40.99
Speaker	73.50	69.39	**26.50**	30.19	58.92	47.57	40.78	76.09	56.03
Table	56.21	47.44	27.45	32.63	41.26	34.78	**27.12**	43.02	40.38
Average	59.20	56.74	29.27	36.92	43.011	33.73	**26.91**	40.36	40.43

Table 4.5 GPOD: Denoising performance evaluated in terms of Mean chamfer distance ($\times 10^{-6}$), for $\sigma = 0.01$. Variance-specific trained network.

Class	Noisy	DGCNN [31]	APSS [47]	RIMLS [46]	AWLOP [49]	MRPCA [53]	GLR [54]	PCN [57]	**GPOD**
Average	59.20	53.14	29.27	36.92	43.011	33.73	26.91	40.36	**24.68**

The denoised point clouds for each method are presented in Fig. 4.4 to provide a visual comparison of their performance at a medium noise level. To better understand the position of the denoised points in relation to the ground truth, the surface distance of each point is also visualized in the figure. The root mean square value of the surface distance (RMSD) can be calculated to further evaluate the performance of each method:

$$\mathrm{RMSD} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \min_{j} ||\hat{\mathbf{x}}_i - \mathbf{x}_j||_2^2}. \tag{4.5}$$

On average, it can be seen that GPOD produces a point-surface distance that is lower, and the reconstructed point cloud is more similar to the original shape compared to the other methods.

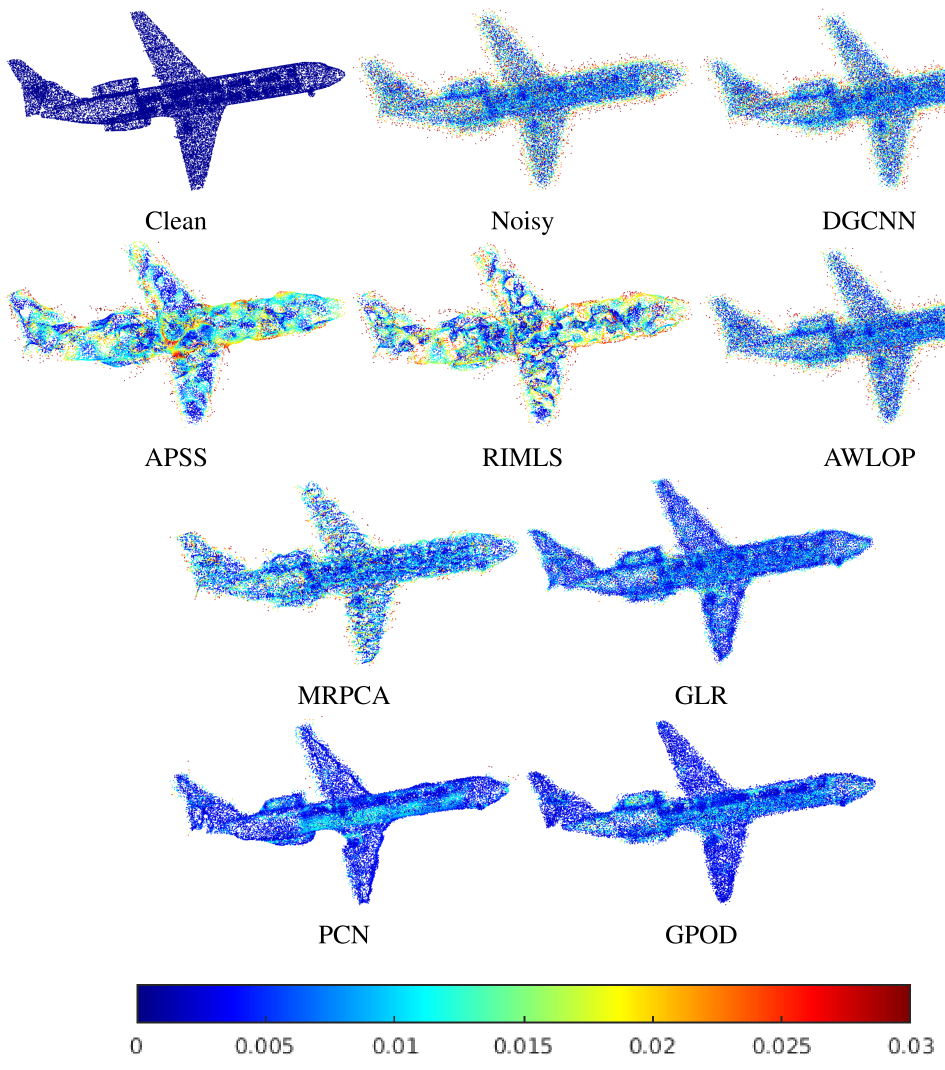


Fig. 4.4 GPOD: Denosing results for $\sigma = 0.015$. Color represents distance to surface (red is high, blue is low). Top left to bottom right: clean point cloud, noisy point cloud, DGCNN (RMSD = 0.0091), APSS (0.0123), RIMLS (0.0127), AWLOP (0.0106), MRPCA (0.0096), GLR (0.0070), PointCleanNet (0.0065), GPOD (0.0062).

4.3.4 Denoising performance with outlier removal

In this section, the performance of denoising methods with outlier removal is evaluated. The comparison is made between various denoising methods that include outlier removal. PointCleanNet [57] and DGCNN [31] are considered as learning-based methods and STM [85] + GLR [54] as a traditional model-based approach. For both PointCleanNet and DGCNN two networks are used, first outlier detection networks are used to identify and remove the outliers, and then the remaining points are fed into the denoising network to obtain the final denoised results. The same thresholds exploited in Sec. 4.3.2 are applied. The chamfer distance is again used to evaluate the denoising performance in the presence of outliers. In this scenario, the number of points in the denoised and original point clouds, N_2 and N_1 in Eq. (4.4), will differ depending on the number of the detected outliers. The median C2C values for the categories in the test set are reported in Tables 4.6, 4.7, and 4.8 for different levels of standard deviation.

Table 4.6 GPOD: Denoising with outlier removal performance evaluated in terms of chamfer distance ($\times 10^{-6}$) for $\sigma = 0.02$.

Class	Noisy	STM + GLR [85] [54]	DGCNN [31]	PCN [57]	**GPOD**
Airplane	2888.50	95.99	132.06	**55.52**	59.80
Bench	2947.12	89.6	122.16	55.53	**50.73**
Car	2508.31	83.35	137.46	133.41	**81.53**
Chair	2381.04	92.90	162.18	108.63	**75.07**
Lamp	2958.91	120.40	194.73	**59.56**	66.29
Pillow	2816.90	91.48	198.14	**56.05**	74.42
Rifle	3958.87	102.18	81.72	49.73	**38.92**
Sofa	2527.52	97.00	162.54	122.23	**82.63**
Speaker	2328.51	87.21	155.96	134.34	**77.36**
Table	2720.44	85.77	144.75	81.58	**52.10**
Average	2803.61	94.59	149.17	85.66	**65.88**

Table 4.7 GPOD: Denoising with outlier removal performance evaluated in terms of chamfer distance ($\times 10^{-6}$) for $\sigma = 0.015$.

Class	Noisy	STM + GLR [85] [54]	DGCNN [31]	PCN [57]	**GPOD**
Airplane	2849.51	40.05	81.18	39.54	**29.60**
Bench	2865.19	37.90	70.71	33.73	**28.01**
Car	2476.96	64.21	95.72	107.50	**59.25**
Chair	2367.17	56.78	110.92	69.48	**48.68**
Lamp	2818.64	45.77	129.65	35.78	**32.36**
Pillow	2799.90	33.37	124.37	51.33	**30.04**
Rifle	3968.17	40.52	44.02	27.99	**24.40**
Sofa	2408.57	60.29	108.48	74.87	**49.91**
Speaker	2323.95	50.08	120.01	85.31	**43.16**
Table	2580.49	42.66	88.69	55.13	**35.81**
Average	2745.86	47.16	97.37	58.07	**38.12**

Table 4.8 GPOD: Denoising with outlier removal performance evaluated in terms of chamfer distance ($\times 10^{-6}$) for $\sigma = 0.01$.

Class	Noisy	STM + GLR [85] [54]	DGCNN [31]	PCN [57]	**GPOD**
Airplane	2765.50	**26.38**	41.50	28.51	33.28
Bench	2790.02	**26.36**	36.32	27.53	34.80
Car	2425.01	**50.87**	63.01	74.52	54.56
Chair	2358.60	**43.66**	63.19	57.44	52.01
Lamp	2830.85	**24.98**	67.12	36.52	33.21
Pillow	2692.34	**19.76**	67.69	24.59	28.85
Rifle	3845.30	23.91	19.92	**17.01**	39.33
Sofa	2431.10	**38.04**	60.89	48.030	42.04
Speaker	2330.73	**37.17**	75.36	51.30	41.54
Table	2610.38	**36.42**	47.88	41.56	39.78
Average	2707.98	**32.75**	54.29	40.70	39.94

Table 4.9 GPOD: Denoising with outlier removal performance evaluated in terms of Mean chamfer distance ($\times 10^{-6}$) for σ = 0.01. Variance-specific training.

Class	Noisy	STM + GLR [85] [54]	DGCNN [31]	PCN [57]	**GPOD**
Average	2707.98	32.75	62.24	48.46	**29.77**

The results show that the proposed method outperforms state-of-the-art denoising methods at medium and high levels of noise, as seen in Tables 4.6 and 4.7. However, at low levels of noise, the model-based approach (STM+GLR) is more effective. Nevertheless, GPOD still outperforms other learning-based methods even being a single model. The reason for this behavior is similar to what was observed in the denoising experiments discussed in section 4.3.3. If more information about the noise is available, the model performance can be improved further by performing a variance-specific training. Specifically, at low levels of noise, our method outperforms STM+GLR, as shown in Table 4.9. Tables 4.10 and 4.11 report the results using the Cloud to Plane metric [86] and exhibit a similar behavior to the results discussed above.

Table 4.10 GPOD: Denoising with outlier removal performance evaluated in terms of Mean Point to Plane distance ($\times 10^{-6}$).

σ	Noisy	STM + GLR [85] [54]	DGCNN [31]	PCN [57]	**GPOD**
0.02	2110.32	61.86	110.2	**41.58**	41.93
0.015	2058.11	24.41	68.39	25.09	**17.32**
0.01	2050.00	**14.39**	35.61	19.80	19.98

Table 4.11 GPOD: Denoising with outlier removal performance evaluated in terms of Mean Point to Plane distance ($\times 10^{-6}$) for σ = 0.01. Variance-specific training.

	Noisy	STM + GLR [85] [54]	DGCNN [31]	PCN [57]	**GPOD**
Average	2050.00	14.39	39.27	22.65	**11.30**

The denoising with outlier removal task qualitative results are presented in Fig. 4.5. The figure shows the denoised point clouds without outliers and the surface distance of each point. To measure the point-surface distance, the RMSD metric introduced earlier for Fig. 4.4 is used. Compared to other methods, GPOD has a lower point-surface distance on average. Furthermore, GPOD is more effective in reconstructing the original shape and surfaces of the point cloud without losing important details or thinning the shape.

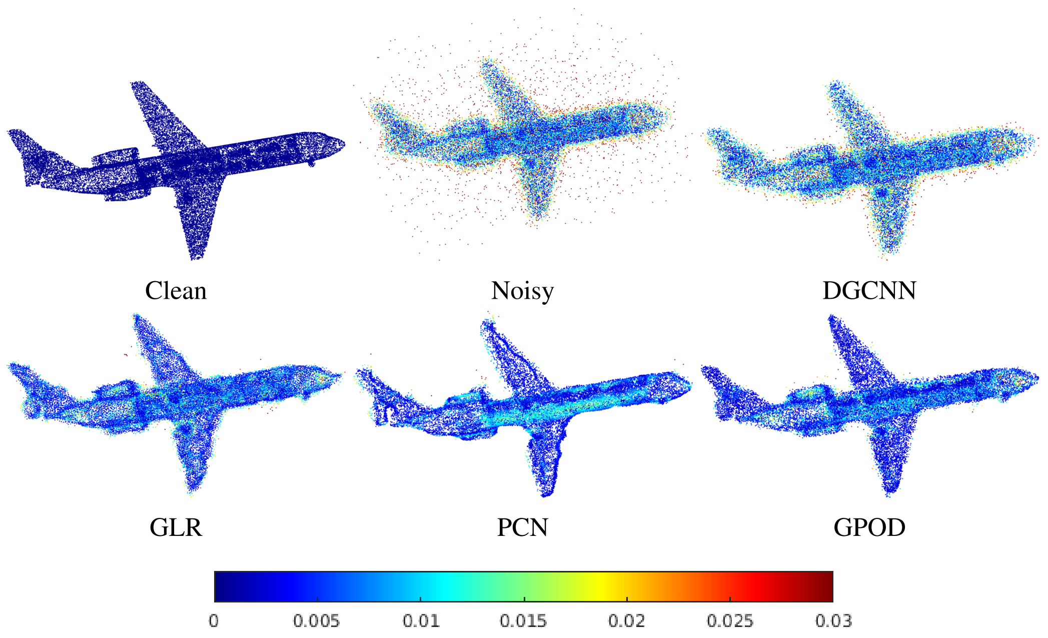


Fig. 4.5 GPOD: Denosing results for $\sigma = 0.015$ and outliers. Color represents distance to surface (red is high, blue is low). Top left to bottom right: clean point cloud, noisy point cloud, DGCNN (RMSD = 0.011), STM + GLR (0.0077), PointCleanNet (0.0063), GPOD (0.0064).

4.3.5 Real noise removal

In order to check if the proposed architecture can generalize beyond white Gaussian noise, the proposed model is tested on two realistic denoising settings. For the experiment the process of scanning Shapenet objects using a Velodyne HDL-64E scanner is simulated via the Blensor software [84]. Two sources of noise are considered: a laser distance bias with Gaussian distribution and per-ray Gaussian noise. Both have zero-mean and a standard deviation equal to 1% of the longest side of the object bounding box. The study also compares the proposed method with PointCleanNet, as representative state-of-the-art model, that has been retrained on the simulated data. Table 4.12 presents the results, which are consistent with those obtained for white

Gaussian noise. The proposed method outperforms PointCleanNet in this case as well. Since the points are not uniformly distributed, RMSD is used instead of the chamfer distance.

Table 4.12 GPOD: Velodyne scan structured noise evaluated in terms of RMSD, network with 8-NN.

Noisy	PCN	**GPOD**
0.1447	0.0966	**0.0602**

In the second experiment, real point clouds generated by image-based 3D reconstruction techniques are used. These point clouds are usually very noisy and contain a lot of outliers due to image imperfections. The multiple-view plane-sweep algorithm is used for the image-based reconstruction method, and a point cloud generated by the algorithm implementation in a previous study is considered for the denoising task. The noisy point cloud is shown in Fig. 4.6 on the left, and it is denoised using GPOD and PointCleanNet. Since the ground truth is not available, qualitative results are reported in Fig. 4.6 as benchmark. GPOD produces a denoised point cloud with fewer diffused outliers than PointCleanNet, as seen in the bottom-right portion of the point clouds in Fig. 4.6, where PointCleanNet presents a cluster of points outside of the main shape of the torch that cannot be easily removed without compromising the entire shape. Overall, GPOD is able to reconstruct sharper object details, as seen in the body of the torches.

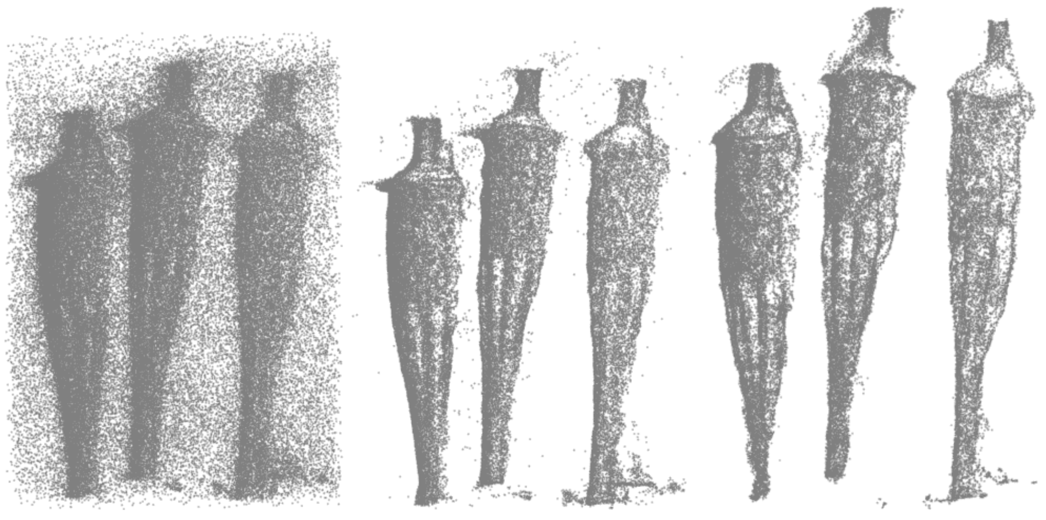


Fig. 4.6 GPOD: Denoising results for real noise.

4.4 Method analysis

4.4.1 Low-rank approximation analysis

In Section 2.2.2, the implementation of the aggregation weight matrix in the graph-convolution operation is described, with an emphasis on enforcing a low-rank approximation of maximum rank r to limit memory and computation complexity, as well as reduce vanishing gradient effects. Several tests were conducted to analyze the network's behavior as a function of the chosen maximum rank, and the results of the denoising with outlier removal and pure denoising tasks are presented in Tables 4.13 and 4.14. The chosen value for the previous experiments, $r = 11$, was compared to a low-rank value (lowest complexity) and the highest rank that fit the GPU memory. Interestingly, it was found that a higher rank does not necessarily lead to better performance. Instead, it was observed that $r = 11$ not only provides a desirable working point in terms of complexity but also achieves the best performance.

Table 4.13 GPOD: Denoising with outlier removal performance. Experiments for maximum rank.

	F1			**C2C** ($\times 10^{-6}$)		
r	0.02	0.015	0.01	0.02	0.015	0.01
3	$89.01^{\pm 0.05}$	$90.30^{\pm 0.06}$	$91.05^{\pm 0.05}$	$69.75^{\pm 0.09}$	$41.56^{\pm 0.31}$	$43.91^{\pm 0.13}$
11	$\mathbf{89.43^{\pm 0.04}}$	$\mathbf{91.37^{\pm 0.05}}$	$\mathbf{92.56^{\pm 0.04}}$	$\mathbf{65.96^{\pm 0.11}}$	$\mathbf{38.20^{\pm 0.09}}$	$\mathbf{39.97^{\pm 0.14}}$
18	$89.22^{\pm 0.04}$	$90.69^{\pm 0.05}$	$91.65^{\pm 0.05}$	$72.03^{\pm 0.30}$	$39.74^{\pm 0.19}$	$40.05^{\pm 0.14}$

Table 4.14 GPOD: Denoising performance. Experiments for maximum rank.

	C2C ($\times 10^{-6}$)		
r	0.02	0.015	0.01
3	$61.75^{\pm 0.05}$	$37.35^{\pm 0.04}$	$41.46^{\pm 0.08}$
11	$\mathbf{61.48^{\pm 0.13}}$	$\mathbf{36.88^{\pm 0.05}}$	$41.47^{\pm 0.04}$
18	$64.21^{\pm 0.07}$	$37.11^{\pm 0.07}$	$\mathbf{39.51^{\pm 0.06}}$

4.4.2 Dynamic graph

This section describes a study on the impact of the graph computation on the performance of the network. Two different approaches are compared: dynamic graph construction, where the graph is computed in the feature space, and fixed graph construction, where the neighbors are identified in the noisy 3-D space. For fixed graph construction, neighbors are computed from the original noisy input for feature extraction and outlier removal blocks, and from the noisy input without the detected outliers for the denoising block. The graph was used for all graph-convolutional layers inside the corresponding block. The results presented in Table 4.15 indicate that the use of dynamic graph update improves the performance of the network due to the ability to find and exploit latent feature correlations.

Table 4.15 GPOD: Fixed vs. Dynamic graph, for $\sigma = 0.015$.

	Denoising with outlier removal		Denoising	
	Dynamic	Fixed	Dynamic	Fixed
C2C ($\times 10^{-6}$)	38.12	53.23	36.78	49.08

4.4.3 Neighborhood size

The effect of neighborhood size is an interesting study to analyze. When a larger number of neighbors is selected for the graph-convolutional layer, it increases the receptive field's size and can assist in denoising smooth areas in the point cloud by capturing more context. However, this comes at the cost of losing some localization and increased computational complexity. This is similar to findings in image denoising, where the optimal receptive field size depends on the noise variance. Tables 4.16 and 4.17 show that increasing the number of neighbors is beneficial up to a certain point and that the optimal number of neighbors actually depends on the noise variance. GPOD faces a tradeoff between good performance at high or low variance since it is a blind method.

Table 4.16 GPOD: Denoising with outlier removal performance. Experiment number of neighbors (NN).

	F1			**C2C** ($\times 10^{-6}$)		
NN	0.02	0.015	0.01	0.02	0.015	0.01
4	86.72	91.08	93.68	81.59	49.12	35.00
8	89.85	91.47	92.45	69.40	41.16	35.62
16	89.43	91.41	92.50	65.88	38.12	39.94
24	89.28	90.98	92.06	61.95	39.09	43.52

Table 4.17 GPOD: Denoising performance. Experiment number of neighbors (NN).

	C2C ($\times 10^{-6}$)		
NN	0.02	0.015	0.01
4	80.18	49.33	34.87
8	65.93	39.80	35.40
16	61.33	36.78	40.43
24	57.27	37.68	44.12

4.5 Conclusion

In this research, we introduce GPOD network, a new graph-convolutional architecture that can effectively identify and remove outlier data points and denoise point cloud data in a single model without prior knowledge of the noise level. The backbone of the network is the graph-convolutional layer that gives the architecture a fully convolutional behavior and enables the network to learn multiple levels of features, similar to a classic CNN. The experimental results show that the proposed network performed well at all noise levels and particularly showed significant improvements over existing methods when dealing with high levels of noise. Additionally, it has been shown that when additional information about the noise is provided, the performance of the network can be further improved.

Chapter 5

Point Cloud Normal Estimation with Graph-Convolutional Neural Networks

5.1 Introduction

One of the main themes of this thesis is to investigate the use of graph convolutional neural networks to learn powerful features from raw point clouds, whose points can be often corrupted by noise. In the research proposed in this chapter, graph convolutional neural networks are explored in a different context with respect to restoration. Here, the ability of learning features able to predict geometric characteristics is studied. In particular, the task of estimation of unoriented surface normals from 3-D point clouds is considered. An in depth description of the task and the relative state-of-the-arts methods can be found in Sec.2.5.3. A graph-convolutional neural network is proposed to estimate unoriented surface normals from raw point clouds. Thanks to graph convolutions, the network can create complex hierarchies of features with a dynamically expanding receptive field, allowing the proposed method to achieve state-of-the-art performance, providing robust estimates even in presence of noise.

5.2 Proposed method

In this section, the proposed network to estimate the normal vector associated to each point of an input point cloud is described.

5.2.1 Architecture

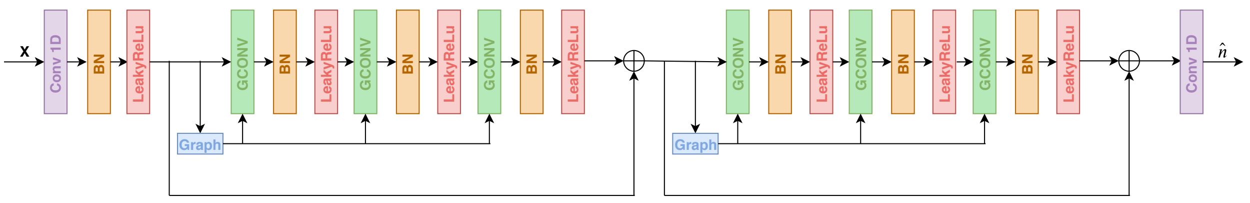


Fig. 5.1 Point Cloud Normal Estimation: Proposed architecture. The network takes as input a patch of N points $\mathbf{x}$ and provides as output the corresponding unoriented normal vectors $\hat{\mathbf{n}}$ per point. The network first projects the noisy point cloud into a feature space by means of one 1-by-1 convolution (Conv1D) and then with several graph convolutional layers (GCONV) it estimates the unoriented normal vectors that are projected back into the 3-D space by a single 1x1 convolutional layer.

An overview of the architecture is shown in Fig. 5.1. The network takes as input a patch of N points of a point cloud and estimates the unoriented normal vector associated to all the points in the input patch.

At high level, the network first project the 3-D input data into the feature space, then several graph convolutional layers are used to uncover latent geometric information of the data to directly estimate the desired geometric feature, the unoriented normal vectors.

The patches are extracted from a point cloud selecting a point and its k-nearest neighbors and they are standardized to have a zero mean and unit standard deviation. Then, the 3-D coordinates of each patch are projected onto an F-dimensional feature space using a single-point convolutional layer, followed by batch normalization and an activation function. The network has two residual blocks, which use skip connections to combine feature vectors from the input and output of the block. Residual connections are well-known for mitigating vanishing gradient problems and improving convergence during training. The residual blocks are responsible for extracting the geometrical information, and each block contains three graph-convolutional layers, each followed by batch normalization and activation functions. The graph convolutional layer exploits the lightweight-ECC, see Sec. 2.2.2 to have more detail. In addition to the feature vectors of the points, the graph-convolutional layer requires a graph that describes connections between points. This graph is constructed dynamically as a nearest neighbor graph using Euclidean distances between the feature vectors of the points at the beginning of each residual block. This dynamic construction helps to promote more powerful feature representations and exploit self-similar patterns. More details of this design choice and the description of the graph convolutional layer can be found in Sec. 3.2. Finally, a single-point convolutional layer is used to project the estimated normals features to the 3-D space after the two residual blocks. This network can estimate the normal for each point in the patch, unlike other methods such as PCPNet [72], which only estimates the central point's normal. However, estimates for points at the patch's edges may not be very accurate due to highly skewed neighborhoods.

5.2.2 Loss Function

For the supervision during training we compute the discrepancy between the predicted normals vectors $\hat{\mathbf{n}}$ aand the actual ground truth normal vectors $\mathbf{n}$. We conducted preliminary experiments with angular losses, such as cosine similarity or angular distance, and Euclidean distance and the latter always proved to be more effective, similar findings are also reported in [72]. The chosen loss function is defined as follow:

$$L = \frac{1}{P} \sum_{i \in \mathbf{S}_P} \min \left(\|\hat{\mathbf{n}}_i - \mathbf{n}_i\|_2^2, \|\hat{\mathbf{n}}_i + \mathbf{n}_i\|_2^2 \right), \tag{5.1}$$

where $\mathbf{S}_P$ is the set of the P closest nodes to the central point of the patch. Only the P closes points to the center are considered in the loss function because nodes that are far from the center of the patch can be affected by border effects that result in highly skewed receptive fields. Therefore, even though our method estimates normals for all points in the patch, we only consider the P nodes closest to the patch center to avoid these border effects. We use more points than just the central point, unlike the PCPNet approach, to improve training efficiency and convergence. Additionally, our method aims to estimate the unoriented normals associated with the points, so we use the minimum function in Eq. (5.1) to select the normal orientation that provides the minimum error for each point.

5.3 Experimental results

In this section, a series of experiments to assess how well the proposed method performs in comparison to model-based baselines and cutting-edge deep learning approaches are conducted. Specifically, we evaluate PCA [67], jet fitting [50], HoughCNN [75], PCPNet [72], and Nesti-Net [73].

5.3.1 Training and testing details

The datasets used for training and testing are the same as those used by PCPNet and Nesti-Net for a fair comparison. The training dataset consists of eight point clouds with 100000 points, which are divided into 10000 patches of 800 points each. During training, the network takes a batch of 16 patches as input and estimates a normal vector for each point in the patch. However, only the normals of the 250 points closest to the center of the patch are used in the loss function, as previously explained.

The proposed method and PCPNet are variance-specific trained. A particular focus is given on their robustness to white Gaussian noise and three different standard deviation levels are considered: 0.012, 0.006, and 0.00125 relative to the bounding box. To construct the graphs for our proposed method, 15 nearest neighbors are used for noiseless and low noise levels, and 35 nearest neighbors for medium-high noise levels. The number of neighbors has been cross-validated on a validation dataset, and the edge attention hyperparameter is set to $\delta = 10$. The network is trained for approximately 100 epochs with an initial learning rate of 10^{-4}, which is then decreased to 10^{-5} after 60 epochs.

The testing dataset consists of 19 point clouds with 100000 points. As in previous works, we choose a subset of 5000 points per point cloud for evaluation of the error metric. For each of these points, a patch of the nearest 800 points is given as input to the network.

5.3.2 Quantitative results

To evaluate how well the proposed method performs, a quantitative measure called the root mean squared (RMS) angle error for unoriented normal estimation is used on 5000 points from the test set. The RMS angle error (measured in radians) for unoriented estimation is calculated as follows:

$$E=\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left[\arccos\left(1-\frac{1}{2}\min(\|\hat{\mathbf{n}}_i-\mathbf{n}_i\|_2^2, \|\hat{\mathbf{n}}_i+\mathbf{n}_i\|_2^2)\right)\right]^2},$$

where $\mathbf{n}_i$ the ground-truth normal vector at point i, $\hat{\mathbf{n}}_i$ the corresponding estimated normal vector and N the number of points.

Table 5.1 presents the RMS angle error achieved by different methods, with the best-performing scale selected for methods with multi-scale variants, for each standard deviation. The proposed method achieves lower average errors and is comparable to or better than state-of-the-art results. Table 5.2 provides further information on the distribution of errors, including the percentiles of the angle error distribution across the entire test set, along with a one-standard-deviation confidence interval. For example, the 99th percentile value of 24.04 degrees indicates that 99% of the points have a normal estimation error below 24.04 degrees. The angle error (in radians) at point i is computed as

$$E_i = \arccos\left(1-\frac{1}{2}\min(\|\hat{\mathbf{n}}_i-\mathbf{n}_i\|_2^2, \|\hat{\mathbf{n}}_i+\mathbf{n}_i\|_2^2)\right).$$

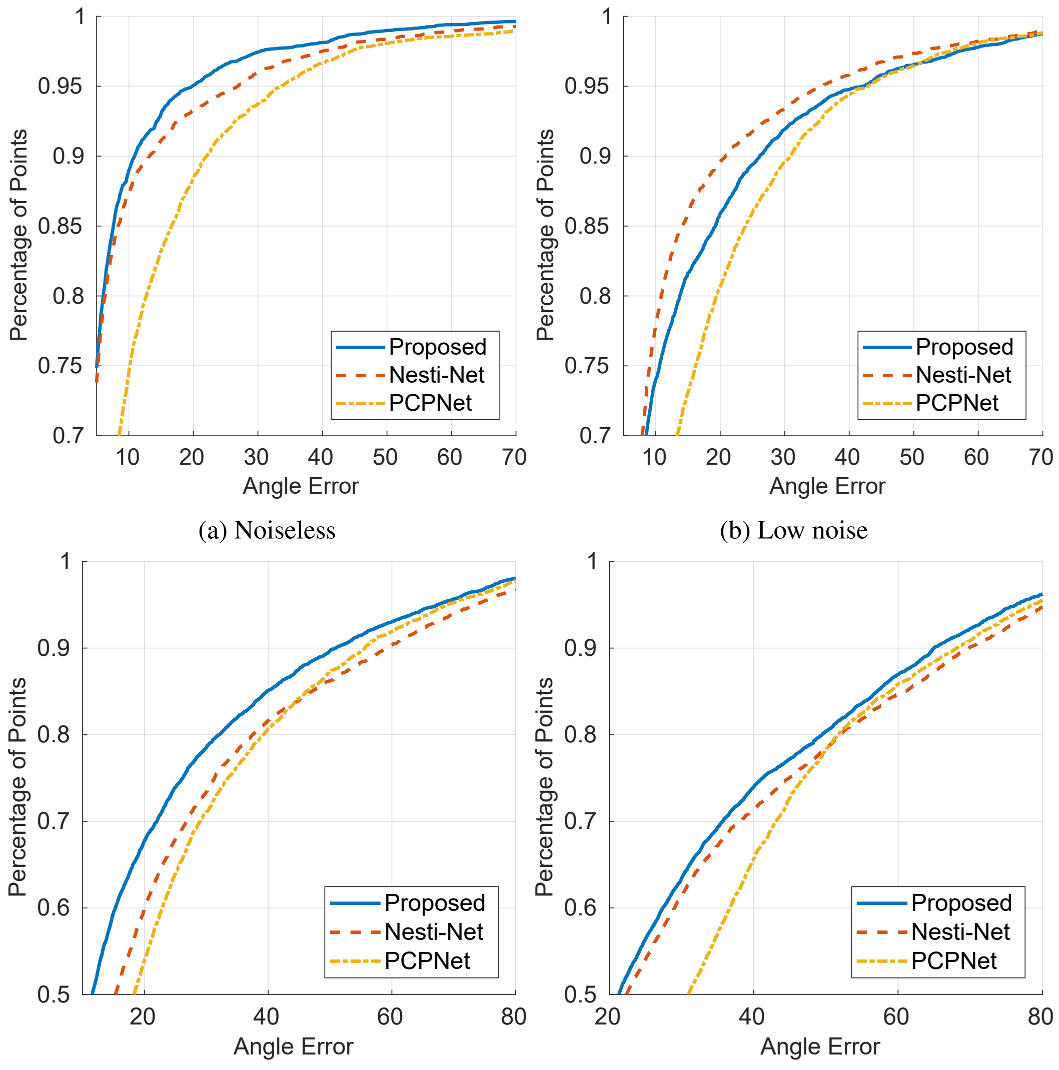
The results indicate that the proposed method has lower error values, which implies that it has fewer points with high normal estimation errors and a shorter tail in the error distribution. This is more desirable than having a lower average error because outliers with high errors can significantly affect the performance of algorithms that rely on high-quality normal estimation. Figure 5.2 shows the high-error tail of the cumulative error distribution for point cloud “star sharp” for the proposed method, Nesti-Net, and PCPNet.

Table 5.1 Point Cloud Normal Estimation: Unoriented RMS angle error (degrees).

	Proposed	Nesti-Net	PCPNet	PCA	Jet	HoughCNN
Noiseless	**6.47**	6.99	8.49	8.31	7.60	10.02
Low Noise	10.73	**10.11**	11.08	12.00	12.36	11.21
Med Noise	**17.53**	17.63	18.26	18.38	18.33	22.66
High Noise	**22.09**	22.28	22.80	23.50	23.41	33.39

Table 5.2 Point Cloud Normal Estimation: Angle error percentiles.

	90 percentile		
	Proposed	Nesti-Net	PCPNet
Noiseless	**8.50**	10.07	12.35
Low noise	15.61	**15.39**	17.28
Med noise	27.17	**26.84**	28.67
High noise	**35.16**	35.17	36.90
	95 percentile		
	Proposed	Nesti-Net	PCPNet
Noiseless	**12.43**	15.13	17.02
Low noise	22.30	**21.46**	22.91
Med noise	**35.37**	35.42	37.19
High noise	**45.57**	45.59	47.33
	99 percentile		
	Proposed	Nesti-Net	PCPNet
Noiseless	**24.04**	25.86	29.66
Low noise	38.34	**34.73**	36.51
Med noise	**56.28**	58.92	57.08
High noise	**65.33**	67.66	66.83



(a) Noiseless (b) Low noise

(c) Medium noise (d) High noise

Fig. 5.2 Point Cloud Normal Estimation: Cumulative distribution of angle errors for point cloud “star sharp”.

5.3.3 Qualitative results

TFig. 5.3 provides a visual representation of the per-point angle error of the estimated normals in comparison to the ground truth for the proposed method, Nesti-Net, PCPNet, and PCA at all noise standard deviations. It is evident from the figure that the proposed method outperforms the other methods, as it has a smaller number of points with high estimation errors. Moreover, the proposed method shows lower errors in the challenging area at the center junction of the star shape, where PCPNet is particularly vulnerable to higher errors.

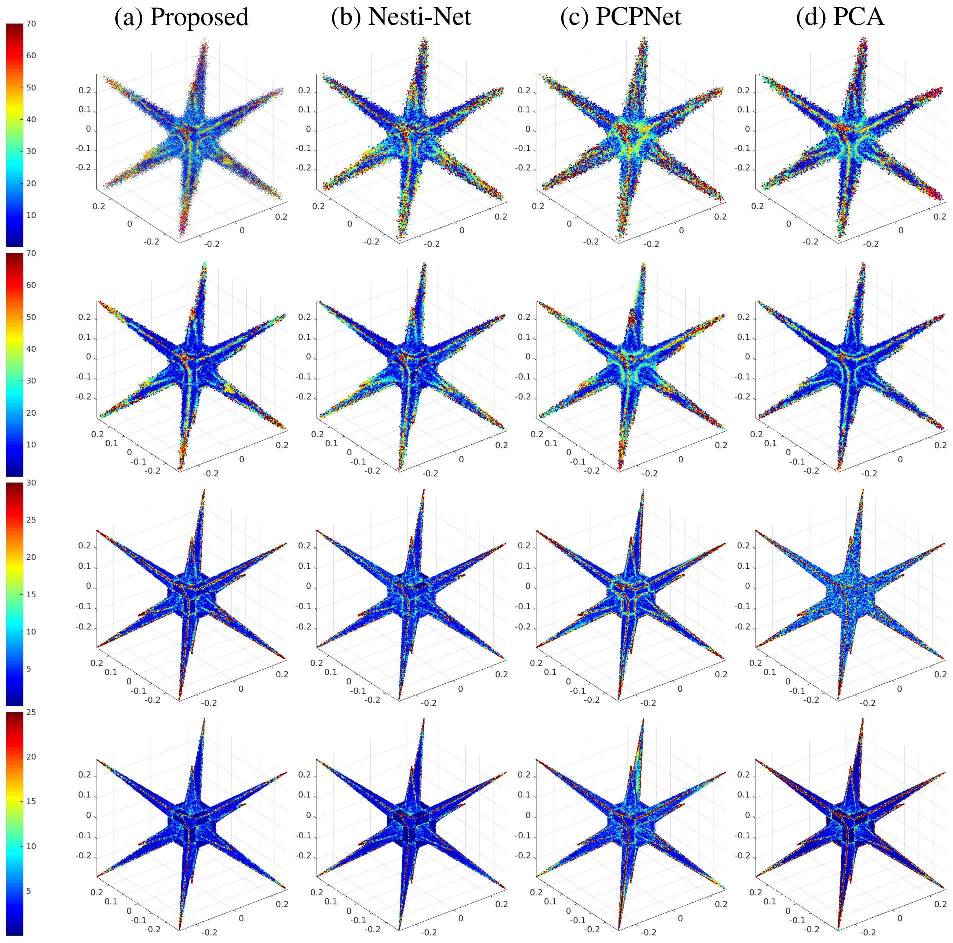


Fig. 5.3 Point Cloud Normal Estimation: Angle errors for point cloud “star sharp” at different level of noise: from a high level of noise (top) to noiseless (bottom).

5.4 Conclusions

A novel approach for estimating surface normals from point clouds using a graph-convolutional neural network is introduced. The results of the proposed method showed significant improvements over existing state-of-the-art techniques, demonstrating its ability to estimate surface normals robustly even in the presence of noise.

Chapter 6

Signal Compression via Neural Implicit Representations

6.1 Introduction

In previous chapters, founding a powerful backbone able tot extract meaningful and robust feature, that is one of the key challenges of point cloud processing, is analyzed for several tasks. Another aspect of paramount importance for point cloud processing is founding efficient and compact point cloud representations, that is analyzed in this chapter.

Although traditional compression methods have relied on model-based techniques, there is a growing interest in using neural networks for compression. These end-to-end compression methods aim to use neural networks throughout the entire compression process without any preconceived notions, allowing for optimal representations to be learned from the data. The most common approach involves using auto-encoder structures[87–92], where an encoder network learns to extract a compact vector from the input signal, and a decoder network produces an estimate of the original signal from the compressed information. This allows for universal encoder and decoder networks, with the compressed information being represented as a quantized and entropy-coded compact vector. While some works have used generative models to obtain compact representations, they are similar in concept to autoencoders. There have also been some attempts to use convolutional dictionary learning and deep learning extensions, but these methods have not been widely applied to compression.

In this research a novel approach for signal compression using neural networks is examined. Instead of using traditional autoencoder structures or dictionary learning, the proposed method uses a neural network that takes one coordinate from the signal domain as input and outputs the corresponding signal value. The network shares parameters across all input coordinates. During training, the network is trained to overfit to the signal, resulting in the network's weights, biases, and architecture serving as a compact representation of the signal. The motivation for this work is given by the recent success of neural implicit representations with periodic activation functions [1] or specific embedding layers [44]. We then present a novel compression paradigm, called NIC (Neural Implicit Compression), that can be applied to any type of signal and we prove its performance on the task of point cloud attribute compression as an example application. In NIC, the network takes in a single coordinate from the signal domain and returns the corresponding value of the signal, while sharing parameters for all input coordinates. The importance of efficient coding of the weights of the network is demonstrated, as the rate of representation depends solely on that. It is discussed the use of priors through meta-learning, which involves a universal pretraining of the network for a given class of signals. The final value of the weights is then obtained through finetuning on the signal of interest, allowing for differential encoding with respect to their universal initializations and significant rate savings compared to random initialization. Furthrmore, the connection between NIC and transform coding are formalized. This study is an initial investigation into a new compression approach, which raises several research questions that can enhance the design. Despite this, the research demonstrates very positive experimental outcomes on a sample application, specifically the compression of point cloud attributes. This application makes full use of the new approach's benefits, as traditional compression methods are challenging to design due to the irregularity of the domain, while our framework can handle it easily and achieve performance similar to the most advanced MPEG G-PCC standard.

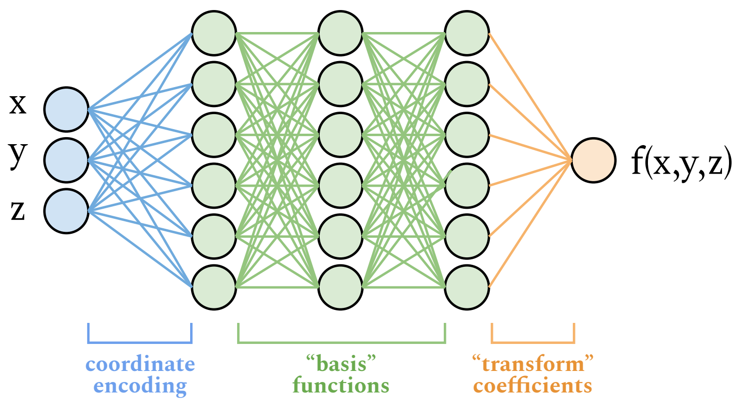


Fig. 6.1 NIC: Connections between transform coding and implicit neural representations.

6.2 Proposed method

6.2.1 Neural Implicit Representations of signals

As described in Sec. 2.3, the idea behind neural implicit representations is to use a neural network, typically a multilayer perceptron (MLP), to represent continuous functions. In this approach, the input to the neural network is a coordinate $\mathbf{x}$ from the domain of the function, and the output is the corresponding value of the function $f(\mathbf{x})$. In other words, as it can be seen from Fig 6.1, the network creates a set of optimized basis functions for the signal and combines them in the final layer, similar to transform coding (see Sec. 6.2.2).

Previous research has mainly used neural implicit representations to solve 3-D rendering problems, but this work is aimed to study their effectiveness as compressed signal representations. It is explored how well these representations can compress signals while maintaining a certain level of accuracy.

In particular, the proposed algorithm is based on SIREN implicit representations [1], which are an MLP with sinusoidal activation functions.

6.2.2 Connections with transform coding

The following result establishes a connection between neural implicit representations and transform coding by demonstrating that there is a direct relationship between the lowest possible value of a two-layer SIREN neural network with enough capacity and the discrete cosine transform (DCT)[93], which is a widely used technique in signal processing.

Proposition 6.2.1. *A two-layer SIREN approximating a continuous function $f : \mathbb{R} \to \mathbb{R}$ with N hidden features is equivalent to an N-point 1D-DCT.*

Proof. Given an input scalar coordinate x, we can define the corresponding output y of the two-layer SIREN as

$$y = \sum_{k=0}^{N-1} w_k^{(2)} \sin(w_k^{(1)} x + b_k^{(1)}) + b_k^{(2)}$$

where $\mathbf{b}^{(i)}$ is the bias of the i-th layer, and $\mathbf{w}^{(1)} \in \mathbb{R}^N$ and $\mathbf{w}^{(2)} \in \mathbb{R}^N$ are the weights of the first and second layer of the network, respectively. Assuming that the input coordinates of the SIREN are regularly sampled on a grid (i.e., $x = n$ where $0 \leq n \leq N-1$), and the loss function is the mean squared error between the output of the network y and the corresponding values of the continuous function $f(x)$ sampled on that grid, it can be shown that the SIREN will converge to the inverse of the N-point 1-D DCT, resulting in the global minimum of the loss function. The 1-D DCT is defined as:

$$y_n = \sum_{k=0}^{N-1} \hat{y}_k \cos\left(\frac{\pi}{N} k \left(n + \frac{1}{2}\right)\right) \quad \text{with } 0 \leq n \leq N-1,$$

where $\hat{y}_k$ is the k-th DCT coefficient of the signal y. Therefore, the two-layer SIREN reaches the global minimum (zero MSE), i.e., converges to the inverse of the N-point 1-D DCT, when $w_k^{(2)} = \hat{y}_k$, $w_k^{(1)} = \frac{\pi}{N} k$, $b_k^{(1)} = \frac{\pi}{2N} k + \frac{\pi}{2}$, and $b_k^{(2)} = 0$. □

This suggests that the initial layer of the neural network generates a set of basis functions, and the final layer learns how to combine them with appropriate weights, similar to transform-domain coefficients. However, it should be noted that this interpretation only holds for shallower networks when fewer features are used.

6.2.3 Signal compression

Compressing a signal with a neural implicit representations requires to perform the following basic operations:

1. Define the architecture of the coordinate-based MLP to be used; this step requires the choice of a network design (e.g, SIRENs), as well as suitable sizing of the model in terms of number of parameters and layers. The size of the model directly affects the accuracy of signal fit, but scaling laws are still unclear (e.g., whether depth or width is more important) as this is the first work looking at the efficiency of such representations. Preliminary empirical evidence from our experiments seems to suggest a preference for width instead of depth. The choice of the network architecture has to be made finding a trade-off between the desired performance and the desired rate requirements. In general, the bigger is the the network the higher accuracy is possible to achieve, also same performance can be achieved by different networks but at different rate, as reported and better described in the esperimental section.

2. Train the network by minimizing a suitable regression loss. For example, denoting as $\mathbf{x}$ a sampled input coordinate and with $y_{\mathbf{x}}$ the signal value at that coordinate, the network parameters θ can be learned by minimizing the MSE:

$$\hat{\theta} = \arg\min_{\theta} \sum_{\mathbf{x}} \|f_\theta(\mathbf{x}) - y_{\mathbf{x}}\|_2^2$$

3. Compactly represent the weights of the network. Since the rate of the compressed representation entirely relies on coding the values of the weights and biases, they must be represented in the most efficient way. Techniques like network sparsification [94] and quantization [95] can be used to reduce the number of parameters and/or their precision. As an example, a uniform scalar quantizer can be applied to the parameters of each layer.

4. Use an entropy encoder on the quantized weights and biases and save any required side information (e.g., sparsification pattern, or quantization step sizes)

Decoding the compressed signal simply amounts to performing a forward pass through the sparse/quantized network, for all the coordinates of interest (e.g., all the pixels in a grid of arbitrary resolution). This “random access” property, where any coordinate can be queried for the corresponding signal value, also implies that decoding at different resolutions or on irregular grids is trivial.

6.2.4 Priors via Meta Learning

Implicit neural networks have the ability to learn any signal, but they cannot use any prior knowledge about the data properties. To overcome this limitation, the authors propose using meta-learning techniques. Meta-learning [96] is a framework used to solve few-shot learning problems by using prior experience to improve future learning performance. In this case, the network is first trained on a collection of data to find a good initialization point that improves the results for unseen data over the same task. The goal is to learn low-level properties of the data, such as smoothness over the domain or common characteristics of a specific class of data. In [97] the authors propose using well-known meta-learning algorithms to learn the initial weight parameters of coordinate-based neural representations, with the aim of speeding up convergence.

The novel idea presented in this research is to utilize a meta-learned initialization to enhance the efficiency of the representation in terms of rate-distortion. For example, consider a simple class of signals, such as images of faces from a public dataset like FFHQ[98]. The meta-learning pretraining on this dataset ideally learns common patterns in the class, such as eyes, nose, or mouth, and provides an initialization that already includes these features. Finetuning the network on a new image would benefit from this informative initialization, and the final values of the network weights would not deviate much from their initial values. To exploit this, a novel compression method is proposed where the difference between the final weights of the network and their meta-learned initialization is quantized and used to recover the signal instead of directly quantizing the final values. This approach achieves significant rate savings because the meta-learned initialization is a strong predictor of the final weight value. Furthermore, the meta-learned initialization is universal and can be created from the public dataset or written in a standard format, so it does not need to be encoded.

There are various meta-learning algorithms[96, 99] described in literature, and in this research a simplified version of Model-Agnostic Meta Learning (MAML) is used. The proposed algorithm operates on one signal at a time, such as an image, which is passed through the coordinate-based network. A few rounds of inner optimization are carried out, followed by an update to the outer optimizer. This process is repeated for each new input signal.

6.3 Experiments

In this section, the effectiveness of the NIC approach is assessed in compressing point cloud attributes, which is used as an illustrative example application. To achieve this, the implicit neural network used is a SIREN that takes as input the 3-D coordinates of a point cloud as input and outputs the corresponding RGB values.

6.3.1 Experimental setting

To prepare the SIREN neural network for point cloud attribute compression, we first selected a suitable design for the number of layers and parameters. We then pre-trained the network using a meta-learning algorithm from Sec. 6.2.4. Specifically, Stochastic Gradient Descend (SGD) with a learning rate of 0.01 as the inner optimizer and Adam with a learning rate of 10^{-5} as the outer optimizer have been used. During pre-training, we ran 10 inner steps for 1000 iterations on a collection of point clouds from the Microsoft Voxelized Upper Bodies dataset [100]. After pre-training, we fine-tuned the network for each point cloud in the test set. We used the Adam optimizer with a fixed learning rate of 10^{-5} for this step.

Each network is trained for 20000 iterations by minimizing a loss function that is based on the MSE between the original colors and the predicted ones in the YUV space:

$$L_{\text{Tot}} = \alpha \text{MSE}_{\text{Y}} + \beta \text{MSE}_{\text{U}} + \gamma \text{MSE}_{\text{V}}, \tag{6.1}$$

where α, β and γ are coefficients aimed modulating the relative importance of luminance and chrominance. In the proposed experiments $\alpha = 0.6, \beta = 0.2, \gamma = 0.2$ are used in order to slightly promote luminance, as common practice in traditional codecs. After finetuning, the network parameters are differentially encoded with respect to the meta-learned intialization. The differences are quantized with a uniform scalar quantizer, where the quantization step size is adapted layer-by-layer on the basis of the dynamic range of the parameters belonging to each layer, and entropy-coded with an arithmetic encoder.

Table 6.1 NIC: BD-Rate over the total PSNR of NIC versus RAHT.

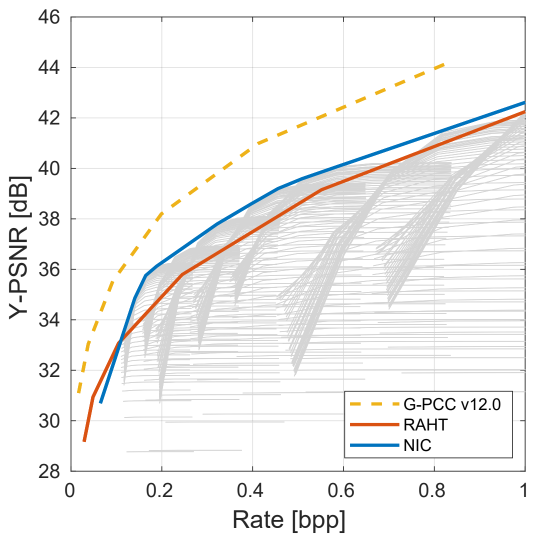
Loot	Longdress	Redandblack	Soldier
-10.23 %	-38.88 %	-22.10 %	-33.39 %

6.3.2 Experimental results

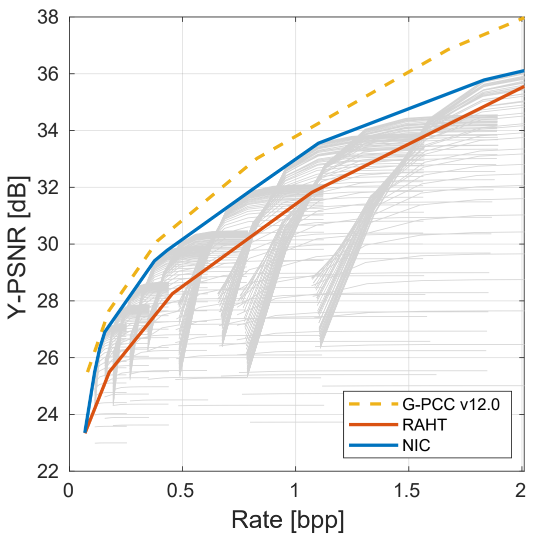
The test set used for compression consists of four point clouds from the 8i Voxelized Full Bodies dataset[101]: `Loot_vox10_1200`, `Longdress_vox10_1300`, `Redandblack_vox10_1550`, `Soldier_vox10_0690`. Notice that these point clouds are different from the data used for meta-learning. To achieve different rate-distortion points, several networks are trained with different numbers of layers, features, and quantization step sizes. In particular, networks with 60, 80, 130 and 170 features per layer, with 5, 7 or 9 hidden layers, and quantization step sizes corresponding to a number of levels from 2^2 to 2^{12} are tested.

The proposed method performance is compared to the latest version of the MPEG G-PCC standard (v12.0 test model) and the Region-Adaptive Hierarchical Transform algorithm (RAHT), a recent algorithm that exploits a hierarchical transform based on Haar wavelets. It i possible to notice from Fig.6.2 that the proposed method significantly improves over RAHT and reaches performance close to G-PCC v12.0, especially at low rates. This is especially significant because the method is not extensively optimized, and there are possibilities for further improvements.

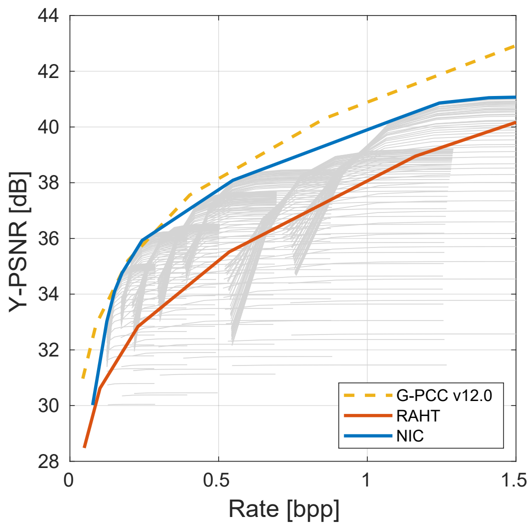
The proposed training method directly promotes the luminance channel over the chrominance, as shown in Eq.(6.1). Total PSNR is also evaluated, and the results reported in Table 6.1 confirm that the proposed method still improves over RAHT even when the distortion on chrominance is taken into account.



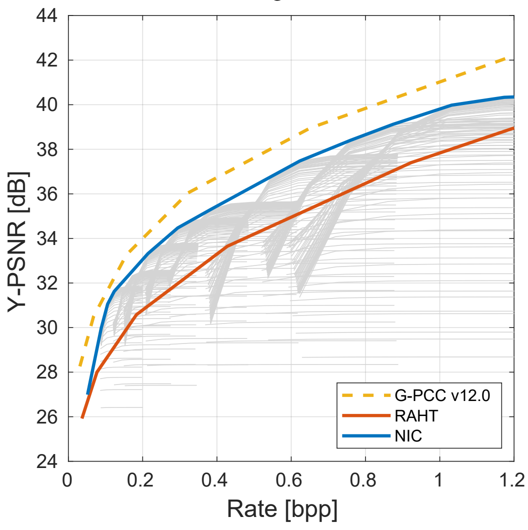
(a) Loot



(b) Longdress



(c) Redandblack



(d) Soldier

Fig. 6.2 NIC: Attribute compression rate-distortion performance. Average BD-Rate: NIC vs. RAHT= -32.07%, NIC vs. G-PCC v12.0 = 47.21%.

6.3.3 Effectiveness of differential meta-learning

In this section the effectiveness of meta-learning is studied by examining the rate-distortion performance achieved by various weight coding options.

First, the distribution of finetuned weights is compared to the distribution of their difference with respect to the meta-learned initialization. The relative results for a network with 80 features and 9 hidden layers trained over the `Redandblack_vox10_1550` point cloud are showed in Fig.6.3. The advantage of differential weight compression is evident, as it results in a distribution with significantly lower variance, leading to increased rate-distortion efficiency.

In Fig.6.4, different methods are compared through rate-distortion curves, including the proposed method with differential compression, classic network compression, i.e., directly quantizing the final values of the weights, and a network with random initialization and quantization of the final values of the weights. The results show that differential coding strategy increases the rate-distortion efficiency, indicating that meta-learning effectively provides prior information about the signal characteristics. However, meta-learning alone without differential compression does not significantly improve performance over random initialization. This is because both networks reach similar quality levels, although they may do so at different rates. Finally, differential coding also has a positive effect on quality, as the distortion introduced on weights is nonlinearly related to the final distortion on the point cloud attributes.

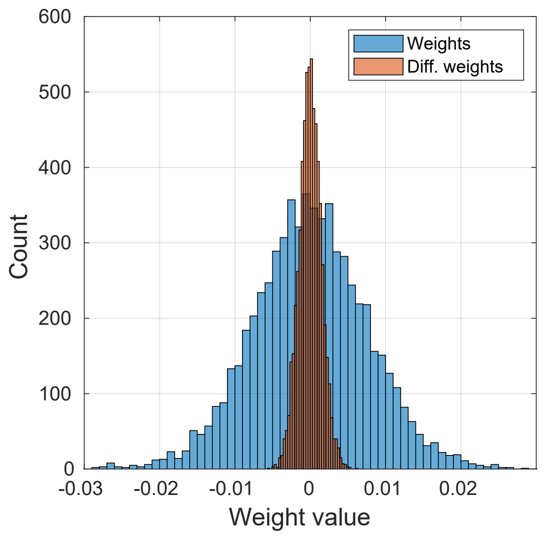
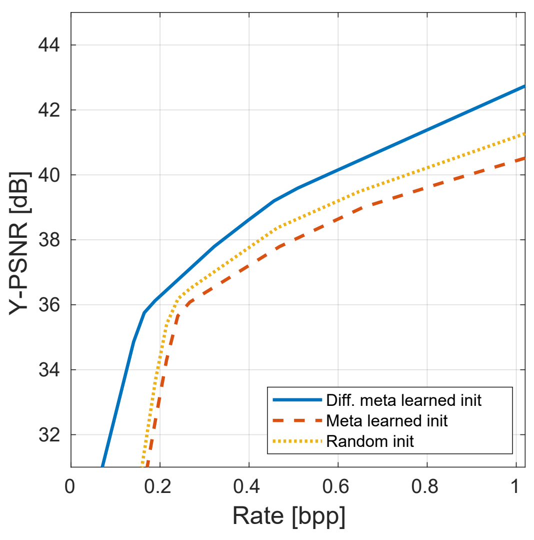
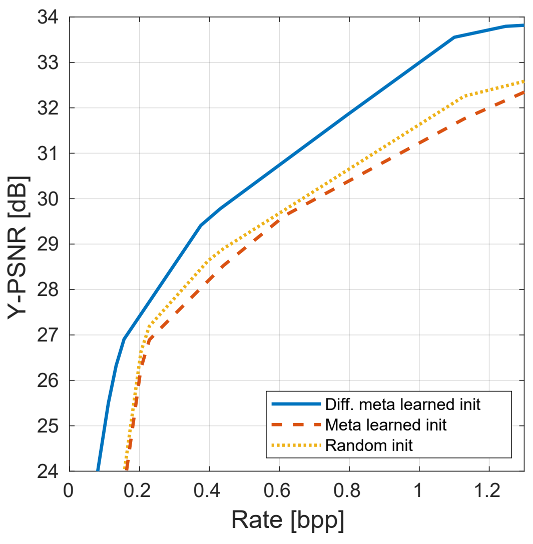


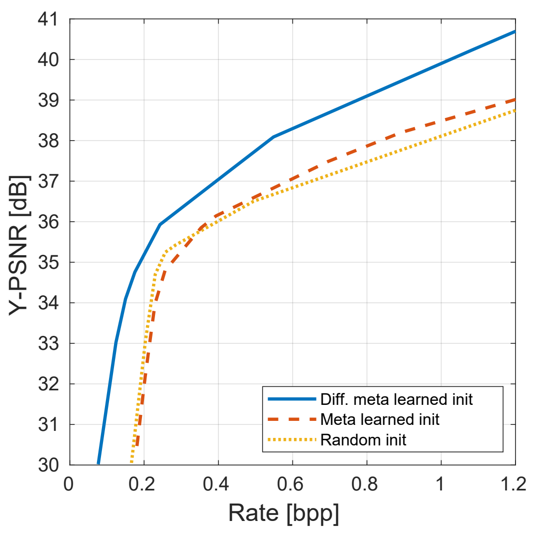
Fig. 6.3 NIC: Distribution of weight values against weight differences.



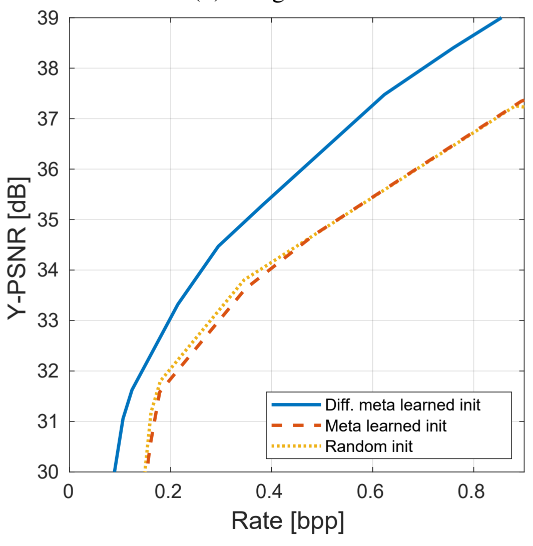
(a) Loot



(b) Longdress



(c) Redandblack



(d) Soldier

Fig. 6.4 NIC: Rate-distortion effectiveness of differential meta-learning.

6.4 Discussion and future developments

This research introduces a new paradigm for compressing signals using neural networks, which involves representing the signal through the weights and biases of a neural implicit representation. This approach is attractive because it can simplify and optimize the design of compression schemes for challenging data types. There are still several open questions regarding this approach that could lead to further research and improved performance. For instance, can the early portion of the network be pre-trained to be more universal, similar to dictionary learning, and potentially save rate? How can network weights be pruned and quantized while maintaining high quality? Can priors be incorporated in new ways besides meta-learning, such as through new architectures that still allow for single-coordinate input?

Chapter 7

Conclusions

In this thesis, the challenges in point cloud processing have been explored focusing on two main themes: learning robust graph convolutional features for point cloud processing in the presence of noise and learning an efficient and compact representation for point cloud attributes.

Firstly, a novel graph convolutional neural network was introduced. The proposed network can effectively learn features from noisy raw data, achieving remarkable results in denoising, surface normal estimation, and outlier removal tasks. The proposed research demonstrated that graph convolutional neural networks are the most promising and powerful tool for point cloud processing tasks, especially in the presence of noisy data.

Related to the second main topic of this dissertation, a novel signal compression algorithm based on neural implicit representations for point cloud attribute compression was proposed. This approach is able to obtain a compact representation of point cloud attributes that can be efficiently compressed, facilitating storage and transmission, improving processing time and overall performance, and reducing costs associated with handling large data sets.

Overall, the results of the presented research show the potential of deep learning and graph-based representations for point cloud processing and signal compression. These techniques have the potential to enable new applications in virtual and augmented reality, robotics, and autonomous systems, and we expect them to play a critical role in the future development of these fields.

7.1 Open problems

Given the rapid progress and increasing interest in the field of point cloud processing, several open problems and challenges remain to be addressed. First of all, the robustness to more complex noise models. In this thesis, the focus was on learning robust features for point cloud denoising in the presence of Gaussian noise. Even if the proposed method has been tested also for real-world simulated noise, the extension to real data is not straightforward. Different and generally difficult to model type of corruption affect real point clouds, such as sensor-specific noise and missing data. Future research could explore the use of graph convolutional networks and test the generalization ability of the model for denoising in these more complex scenarios.

Moreover, real point clouds acquisition are usually characterized by a large number of points. The experiments in this thesis were conducted on relatively small point clouds due to computational constraints. However, as larger point clouds become more prevalent, it will be important to investigate methods for effective scaling of graph convolutional networks.

Furthermore, the compression efficiency of point cloud attributes can be improved. The proposed compression algorithm for point cloud attribute compression is a proof of concept of a generic paradigm that can be further improved and adapted to the specific chosen task. Finally, a joint compression of point cloud geometry and attributes can be investigated. Most existing compression techniques for point clouds either focus on compressing the geometry or the attributes separately. A promising research direction is to develop joint compression techniques that can compress both the geometry and attributes of a point cloud simultaneously, while maintaining their interdependence.

References

[1] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

[2] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

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

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

[5] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5010–5019, 2018.

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

[7] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 922–928. IEEE, 2015.

[8] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition, pages 652–660, 2017.

[9] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[10] Francesca Pistilli, Giulia Fracastoro, Diego Valsesia, and Enrico Magli. Learning graph-convolutional representations for point cloud denoising. In *ECCV*, 2020.

[11] Francesca Pistilli, Giulia Fracastoro, Diego Valsesia, and Enrico Magli. Learning robust graph-convolutional representations for point cloud denoising. *IEEE Journal of Selected Topics in Signal Processing*, 15:402–414, 2021.

[12] Francesca Pistilli, Giulia Fracastoro, Diego Valsesia, and Enrico Magli. Point cloud normal estimation with graph-convolutional neural networks. In *2020 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, pages 1–6, 2020.

[13] Francesca Pistilli, Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Signal compression via neural implicit representations. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3733–3737, 2022.

[14] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009.

[15] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In *2008 IEEE/RSJ international conference on intelligent robots and systems*, pages 3384–3391. IEEE, 2008.

[16] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent point feature histograms for 3d point clouds. In *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany*, pages 119–128, 2008.

[17] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: science and systems*, volume 2, page 435. Seattle, WA, 2009.

[18] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling*, pages 145–152. IEEE, 2001.

[19] Chavdar Papazov and Darius Burschka. An efficient ransac for 3d object recognition in noisy and occluded scenes. In *Computer Vision–ACCV 2010: 10th Asian Conference on Computer Vision, Queenstown, New Zealand, November 8-12, 2010, Revised Selected Papers, Part I 10*, pages 135–148. Springer, 2011.

[20] Richard Vock, Alexander Dieckmann, Sebastian Ochmann, and Reinhard Klein. Fast template matching and pose estimation in 3d point clouds. *Computers & Graphics*, 79:36–45, 2019.

[21] Bing Jian and Baba C Vemuri. Robust point set registration using gaussian mixture models. *IEEE transactions on pattern analysis and machine intelligence*, 33(8):1633–1645, 2010.

[22] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[23] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[24] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[25] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[27] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[28] 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)*, pages 29–38, July 2017.

[29] Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2598–2606, 2018.

[30] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

[31] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):146, 2019.

[32] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10296–10305, 2019.

[33] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

[34] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Deep graph-convolutional image denoising. *IEEE Transactions on Image Processing*, 29:8226–8237, 2020.

[35] Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. pages 9203–9211, 06 2019.

[36] Lars M. Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. *CoRR*, abs/1812.03828, 2018.

[37] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[38] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[39] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4857–4866, 2020.

[40] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *Proceedings IEEE International Conf. on Computer Vision (ICCV)*, 2019.

[41] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[42] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[43] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Deformable neural radiance fields. *arXiv preprint arXiv:2011.12948*, 2020.

[44] Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33, 2020.

[45] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics*, 9(1):3–15, 2003.

[46] A Cengiz Öztireli, Gael Guennebaud, and Markus Gross. Feature preserving point set surfaces based on non-linear kernel regression. In *Computer Graphics Forum*, volume 28, pages 493–501. Wiley Online Library, 2009.

[47] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. In *ACM Transactions on Graphics (TOG)*, volume 26, page 23. ACM, 2007.

[48] Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. Parameterization-free projection for geometry reconstruction. In *ACM Transactions on Graphics (TOG)*, volume 26, page 22. ACM, 2007.

[49] Hui Huang, Shihao Wu, Minglun Gong, Daniel Cohen-Or, Uri Ascher, and Hao Richard Zhang. Edge-aware point set resampling. *ACM transactions on graphics (TOG)*, 32(1):9, 2013.

[50] Frédéric Cazals and Marc Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Computer Aided Geometric Design*, 22(2):121–146, 2005.

[51] Haim Avron, Andrei Sharf, Chen Greif, and Daniel Cohen-Or. l1-sparse reconstruction of sharp point set surfaces. *ACM Transactions on Graphics (TOG)*, 29(5):135, 2010.

[52] Yujing Sun, Scott Schaefer, and Wenping Wang. Denoising point sets via l0 minimization. *Computer Aided Geometric Design*, 35:2–15, 2015.

[53] Enrico Mattei and Alexey Castrodad. Point cloud denoising via moving RPCA. In *Computer Graphics Forum*, volume 36, pages 123–137. Wiley Online Library, 2017.

[54] Jin Zeng, Gene Cheung, Michael Ng, Jiahao Pang, and Cheng Yang. 3D point cloud denoising using graph Laplacian regularization of a low dimensional manifold model. *arXiv preprint arXiv:1803.07252*, 2018.

[55] Chinthaka Dinesh, Gene Cheung, and Ivan V Bajic. 3D Point Cloud Denoising via Bipartite Graph Approximation and Reweighted Graph Laplacian. *arXiv preprint arXiv:1812.07711*, 2018.

[56] Yann Schoenenberger, Johan Paratte, and Pierre Vandergheynst. Graph-based denoising for time-varying point clouds. In *2015 3DTV-Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4. IEEE, 2015.

[57] Marie-Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, Niloy J Mitra, and Maks Ovsjanikov. POINTCLEANNET: Learning to Denoise and Remove Outliers from Dense Point Clouds. In *Computer Graphics Forum*. Wiley Online Library, 2019.

[58] Pedro Hermosilla, Tobias Ritschel, and Timo Ropinski. Total Denoising: Unsupervised Learning of 3D Point Cloud Cleaning. *arXiv preprint arXiv:1904.07615*, 2019.

[59] Chaojing Duan, Siheng Chen, and Jelena Kovacevic. 3D Point Cloud Denoising via Deep Neural Network Based Local Surface Estimation. In *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8553–8557. IEEE, 2019.

[60] Riccardo Roveri, A Cengiz Öztireli, Ioana Pandele, and Markus Gross. Pointpronets: Consolidation of point clouds with convolutional neural networks. In *Computer Graphics Forum*, volume 37, pages 87–99. Wiley Online Library, 2018.

[61] Shitong Luo and Wei Hu. Differentiable manifold reconstruction for point cloud denoising. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, page 1330–1338, New York, NY, USA, 2020. Association for Computing Machinery.

[62] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[63] Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. Deformable shape completion with graph convolutional autoencoders. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1886–1895, 2018.

[64] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Learning Localized Generative Models for 3D Point Clouds via Graph Convolution. In *International Conference on Learning Representations (ICLR) 2019*, 2019.

[65] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, volume 36, pages 301–329. Wiley Online Library, 2017.

[66] Henri Gouraud. Continuous shading of curved surfaces. *IEEE transactions on computers*, 100(6):623–629, 1971.

[67] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. *Surface reconstruction from unorganized points*, volume 26. ACM, 1992.

[68] David Levin. The approximation power of moving least-squares. *Mathematics of computation*, 67(224):1517–1531, 1998.

[69] Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999.

[70] Quentin Mérigot, Maks Ovsjanikov, and Leonidas J Guibas. Voronoi-based curvature and feature estimation from point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 17(6):743–756, 2010.

[71] Tamal K Dey and Samrat Goswami. Provable surface reconstruction from noisy samples. *Computational Geometry*, 35(1-2):124–141, 2006.

[72] Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J Mitra. PCPNet learning local shape properties from raw point clouds. In *Computer Graphics Forum*, volume 37, pages 75–85. Wiley Online Library, 2018.

[73] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. Nesti-Net: Normal Estimation for Unstructured 3D Point Clouds using Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10112–10120, 2019.

[74] Janghun Hyeon, Weonsuk Lee, Joo Hyung Kim, and Nakju Doh. NormNet: Point-wise normal estimation network for three-dimensional point cloud data. *International Journal of Advanced Robotic Systems*, 16(4):1729881419857532, 2019.

[75] Alexandre Boulch and Renaud Marlet. Deep learning for robust normal estimation in unstructured point clouds. In *Computer Graphics Forum*, volume 35, pages 281–290. Wiley Online Library, 2016.

[76] Alexandre Boulch and Renaud Marlet. Fast and robust normal estimation for point clouds with sharp features. In *Computer graphics forum*, volume 31, pages 1765–1774. Wiley Online Library, 2012.

[77] Yiqun Xu, Wei Hu, Shanshe Wang, Xinfeng Zhang, Shiqi Wang, Siwei Ma, Zongming Guo, and Wen Gao. Predictive generalized graph fourier transform for attribute compression of dynamic point clouds. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(5):1968–1982, 2020.

[78] Cha Zhang, Dinei Florencio, and Charles Loop. Point cloud attribute compression with graph transform. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 2066–2070. IEEE, 2014.

[79] Cha Zhang, Dinei Florêncio, and Charles Loop. Point cloud attribute compression with graph transform. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 2066–2070, 2014.

[80] Robert A Cohen, Dong Tian, and Anthony Vetro. Attribute compression for sparse point clouds using graph transforms. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1374–1378. IEEE, 2016.

[81] Ricardo L. de Queiroz and Philip A. Chou. Compression of 3d point clouds using a region-adaptive hierarchical transform. *IEEE Transactions on Image Processing*, 25(8):3947–3956, 2016.

[82] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, July 2017.

[83] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[84] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. BlenSor: Blender Sensor Simulation Toolbox. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Song Wang, Kim Kyungnam, Bedrich Benes, Kenneth Moreland, Christoph Borst, Stephen DiVerdi, Chiang Yi-Jen, and Jiang Ming, editors, *Advances in Visual Computing*, pages 199–208, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[85] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3D Point Cloud Based Object Maps for Household Environments. *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge in Robotics)*, 56(11):927–941, 30 November 2008.

[86] Dong Tian, Hideaki Ochimizu, Chen Feng, Robert Cohen, and Anthony Vetro. Geometric distortion metrics for point cloud compression. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3460–3464. IEEE, 2017.

[87] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. In *International Conference on Learning Representations*, 2016.

[88] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018.

[89] David Minnen, Johannes Ballé, and George Toderici. Joint autoregressive and hierarchical priors for learned image compression. *arXiv preprint arXiv:1809.02736*, 2018.

[90] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. Dvc: An end-to-end deep video compression framework. In *Proceedings*

of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[91] Chunlei Cai, Li Chen, Xiaoyun Zhang, and Zhiyong Gao. End-to-end optimized roi image compression. *IEEE Transactions on Image Processing*, 29:3442–3457, 2020.

[92] Maurice Quach, Giuseppe Valenzise, and Frederic Dufaux. Folding-based compression of point cloud attributes. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3309–3313, 2020.

[93] Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. Discrete cosine transform. *IEEE transactions on Computers*, 100(1):90–93, 1974.

[94] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[95] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[96] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017.

[97] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *CVPR*, 2021.

[98] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405, 2019.

[99] Alex Nichol, Joshua Achiam, and J. Schulman. On first-order meta-learning algorithms. *ArXiv*, abs/1803.02999, 2018.

[100] C. Loop, Q. Cai, S.O. Escolano, and Philip A. Chou. Microsoft voxelized upper bodies – a voxelized point cloud dataset. In *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m38673/M72012*, May 2016.

[101] Eugene d'Eon, Bob Harrison, Taos Myers, and Philip A. Chou. 8i voxelized full bodies - a voxelized point cloud dataset. In *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006*, January 2017.