GraphPointNet: Graph Convolutional Neural Network for Point Cloud Denoising

BY

FRANCESCA PISTILLI
B.S, Politecnico di Torino, Turin, Italy, 2017

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2020

Chicago, Illinois

Defense Committee:

Dan Schonfeld, Chair and Advisor
Rashid Ansari
Maurizio Martina, Politecnico di Torino
Enrico Magli, Politecnico di Torino

ACKNOWLEDGMENTS

I would like to thank my family, whose support and affection gave me a solid and unwavering pillar on which shaping my future. The most sincere gratitude is also devoted to my friends, who have supported and helped me during these most exciting and more troubling times in Chicago and these years of study.

FP

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF TABLES (continued)

TABLE **PAGE**

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF ABBREVIATIONS

PC	Point Cloud
NN	Neural Network
CNN	Convolutional Neural Network
DPN	Deep Neural Network
ECC	Edge Conditioned Convolution
GPN	Graph Point Net

SUMMARY

The proposed project is focused on developing a novel neural network for point cloud denoising based on graph convolution operations.

A point cloud is a representation of an object composed of a collection of points expressed in 3-D space coordinates. This type of data is usually acquired by radars, lasers, sensors, electro-optical systems or by reconstruction of 2-D images: all these acquisition methods lead to point clouds typically affected by noise. The project aims to design a network able to efficiently reproduce cleaned 3-D point clouds from noisy observations.

The denoising task is a typical problem addressed in image processing and the current state-of-the-art are Convolutional Neural Networks (CNN), that lead to promising results. The idea developed in this project is to exploit a deep neural network structure composed of convolutional layers also for point cloud denoising.

The novelty of the project is the introduction of a graph-convolutional layer, that implements the Edge-Conditioned-Convolution [1](ECC) to perform a graph-convolution operation over point clouds.

A graph is a generic collection of nodes and edges; in the case under study a graph is build for each point cloud where the nodes are the points of the cloud and the edges are weighted connections between them. The ECC is performed in a dedicated deep neural network, where the feature vector associated to one node at layer $l + 1$ is computed as a weighted local aggregation of the feature vectors at layer l of the node itself and its k-nearest-neighbors points.

SUMMARY (continued)

Several works are available in literature for point cloud denoising; traditional methods are geometrical algorithms, that can be based on local or non-local operators. Some widely used techniques involve the estimation of surfaces of point clouds from noisy observations, others exploit the graph representation of point clouds and apply graph-regularization methods. All these approaches address the task as a classic optimization problem.

Recently, due to the increasing interest in point clouds and point cloud denoising, new approaches have been explored. In particular, several neural networks able to outperform the traditional methods have been published.

None of these networks exploits a graph representation of the data, neither a convolutional structure, proposing instead a quite simple architecture, based on fully connected layer and max-pooling.

To our knowledge, the network proposed in this thesis, called GraphPointNet, is the first neural network based on convolution able to successfully denoise point clouds.

First of all, an introduction of the basic concepts of neural network and graph theory together with the current state-of-the-art are presented. Then the development of the project from the creation of the dataset to the presentation of the architecture is described and analyzed in details.

Finally, the performance evaluation of the proposed network is reported. Quantitative and qualitative tests are performed in order to evaluate the quality of the obtained results. It is shown that the proposed method is able to outperform or at least match the current state-of-the-art.

CHAPTER 1

INTRODUCTION

In the introduction chapter the background concepts of the project are presented: first a brief description of the input data then an overview of neural network and graph theory are given.

1.1 Point Clouds

3-D point clouds are collections of data points that may represent cities, environments, artificial systems or objects of any dimensions. The data are collection of points expressed in 3-D space coordinates (x, y, z), sampled from the surface of the analyzed shape.

Point clouds are becoming increasingly popular thanks to the ability to provide a detailed representation of the real world and the wider use in different areas, such as architecture, medical imaging, virtual reality, aeronautics applications and so on.

Recently, there has been a growing interest regarding the acquisition and processing of point clouds: new techniques to increase the quality of the data and different applications have been investigated.

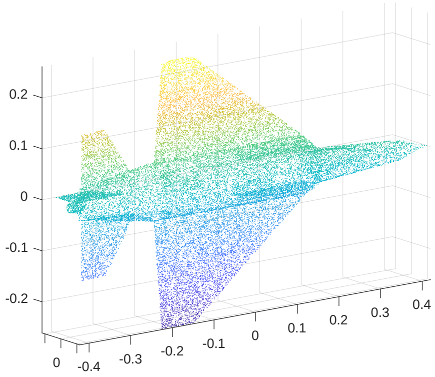


Figure 1: Example of a Point Cloud.

Various approaches to the point clouds acquisition are presented in literature as electro-optical systems, laser scanning, sensors, radars, reconstruction starting from 2-D images.

1.2 Neural Networks

In this section the basic concepts of neural network are presented to provide a general background.

The basic element of a neural network is a *neuron*; it is a mathematical function that emulates the behaviour of biological neurons: several signals arrive at a nerve cell, where they are processed and a response is eventually produced.

A neuron is an algorithm, that takes as inputs vectors of numbers, weights them element by element, sums them together and applies to the results a nonlinear function to obtain the output. The output is a binary classifier, able to make a decision, True or False, according to the weighted input. An example of an artificial neuron is reported in Figure 2.

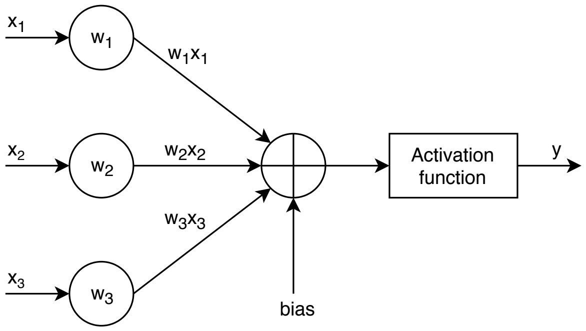


Figure 2: Neuron.

In Figure 2 it can be seen that weighted inputs are summed together with a bias, a scalar quantity inserted to move the decision threshold far from the origin.

The artificial neuron was originally designed for binary classification tasks, as image recognition, and the nonlinear function, also called *activation function*, was a simple function with only two possible outputs, 0 or 1, i.e. True or False. Afterwards, neural networks have been exploited to perform more complex tasks and particularly to approximate any function able to map inputs to the correct outputs. The operations and therefore the structures became gradually more complex.

If several artificial neurons are connected together a neural network is created, see Figure 3 as example. A network can be more or less deep according to the number of hidden layers inserted; a layer can be *fully connected*, if each output of intermediate layers is an input to all neurons of the following layer. How many layers and how to connect them are design decisions, made taking into account the specific task of the network.

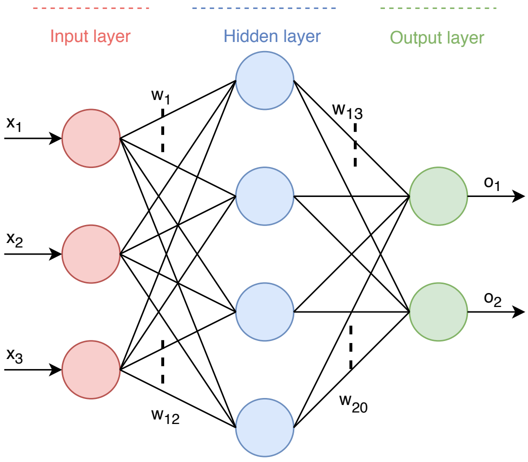


Figure 3: Example of a neural network with one hidden layer.

During the development of a neural network, after the design of the general structure of the net, the weights and the bias are initialized, typically to random values, and the network starts to learn the necessary values to obtain the desired results.

Neural networks are generally characterized by three phases: training, validation and testing.

The training phase is the learning process where the parameters are optimized in order to obtain the desired functionality.To evaluate the network predictions a *loss function* is defined, able to describe the discrepancy between the output of the net and the desired target output. A common function is the mean squared error (MSE), which computes the squared difference between the predicted and the true value:

$$MSE = \frac{1}{N}\sum_{i}^{N}(y - \hat{y})^2, \tag{1.1}$$

where N is the total number of elements, y is the target output and $\hat{y}$ is the output of the net.

It is straightforward to understand that the goal of the training phase is to minimize the loss function: to fulfill this task an optimization method is applied to the loss function in order to optimize the trainable parameters of the network. A widely used optimization algorithm is the gradient descent, that is able to iteratively find a local minimum computing the partial derivatives of the loss function with respect to each trainable variable.

Considering the variable w_1 of the net reported in Figure 3 the variable update would be:

$$w_1 \leftarrow w_1 - \eta\frac{\partial L}{\partial w_1}, \tag{1.2}$$

where L is the loss function and η is the learning rate, a parameter that controls how much the variables can change at each update.

Decreasing the learning rate leads to a slower but more stable learning of the parameters; otherwise increasing it leads to a more oscillating and non-monotonic minimization of the loss function.

During the training phase, a specific training dataset is exploited; a dataset is a collection of data together with the correspondingly true information that has to be predicted. For instance, if the network addresses an image classification task, the dataset would be a collection of images, each associated to a category, usually called *label*, that is the information that has to be predicted. Instead if the network is focused on image denoising task, the dataset would be composed of noisy images and their clean version as label, as the goal of the network is to reconstruct the original clean images. A learning task where the ground truth is available, i.e. the true information that has to be predicted, is called *supervised learning*; otherwise it is defined *unsupervised.*

The validation phase consists in momentarily stopping the training and evaluate the net as far as trained over a validation set. The validation set is composed of data belonging to the same original dataset from which the training dataset is extrapolated but not included in the latter one to ensure that the net has never seen the validation data during the learning phase. The validation phase is performed in order to check if the network is over-adapting the parameters to the training dataset, i.e. the noticed decrease of the loss is due to an over fitting and the same performances are not met with unknown inputs. Notice that only during the training phase the parameters are updated, whereas the validation is an off-running check of the performance. Finally the testing is performed when the network is trained and completely different data are employed.

1.3 Signal Processing application of Neural Network: Convolutional Neural Network

A *Convolutional Neural Network* (CNN) is a class of deep neural networks that has become extremely popular for image processing tasks such as image recognition, classification and denoising. In this section a description of the architecture and the operations involved is provided.

A CNN is a deep neural network that exploits the convolution operation instead of simple matrix multiplication between input and weights. For instance, it takes images as input, each one represented by a tensor with dimensions $(height \times width \times channels)$, where the parameter $channels$ is the depth of the data (one for gray-scale or to three for RGB images) and returns as output the category associated to what is represented in each image.

The internal structure is characterized by several hidden layers, consisting of convolutional layers, followed by activation functions and other additional layers such pooling or fully connected layers.

The main block of the net is the convolutional layer that convolves the input matrix with convolutional filters. A convolutional filter is a matrix of dimension $(height_{filter} \times width_{filter} \times channels)$, where the parameter channels has to be equal to the image's depth, and all the elements of the filters are the trainable weights of the network.

The network is able to detect and identify information and features from the data that can be used to fulfill the addressed task; to do so the input data are convolved by specific filters in order to extract important characteristics. Each neuron can only see and process as input a portion of data, defined as receptive field, to predict the output. If several fields are considered it is possible to cover the whole area of interest.

To perform a convolution operation a filter slides over the input matrix, isolating windows of data, and linearly combines the selected elements with the filter values. A fixed number of shifts is performed to slide the window over the whole matrix and the number of shifts, called *strides*, is set during the design. During the training phase, the network learns and optimizes the filter's elements in order to extract information meaningful for the established task. An example to clarify the operations of a generic CNN is reported in Figure 4.

One advantage of the architecture concerns the decrease of the learning complexity, due to the reduction of the number of parameters involved and the reuse of weights with respect to a network with only fully connected layers.

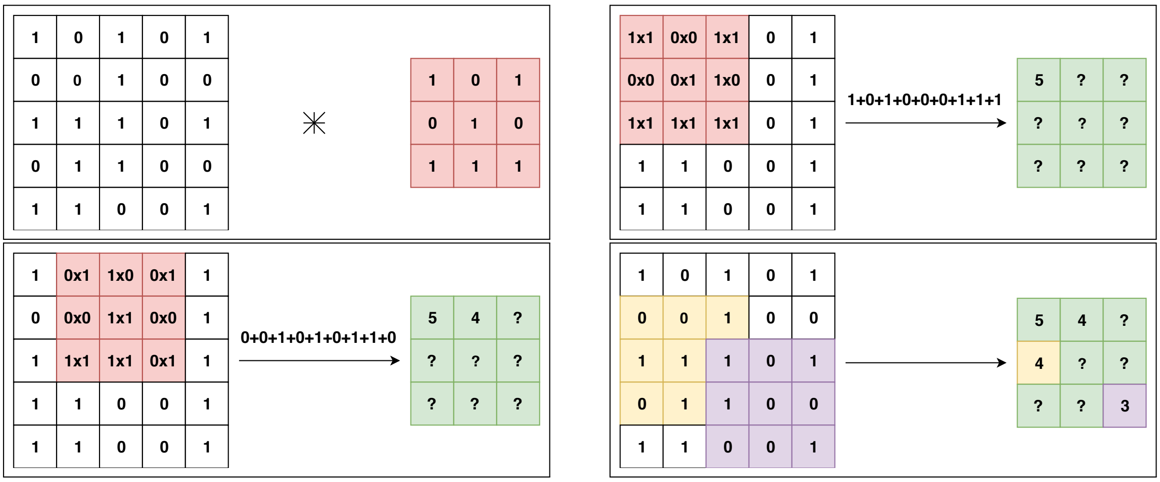
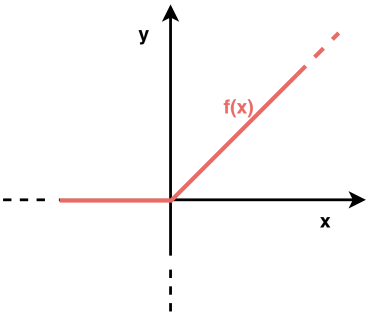


Figure 4: Convolution operation example. Input matrix 5x5 and filter 3x3 (top left), computation of the element (1,1) (top right), of the element (1,2) (bottom left), of elements (2,1) and (3,3) (bottom right). The computations reported regard a convolution operation between the input matrix and the filter in the top left with slide equal to 1.

An activation function is inserted after each convolution layer; one popular non-linear function exploited is the *Rectified Linear Unit* (ReLu):



$$f(x) = \max(0, x). \tag{1.3}$$

Furthermore, a *pooling* layer is often inserted in a convolutional network architecture: this type of layer is capable of reducing the number of parameters, taking into account only the most significant ones by spatially downsampling the feature maps. For instance, a common choice is to perform a *Max Pool*: only the maximum parameter over a selected window is retained.

In Figure 5 an example of a CNN for a classification problem is reported.

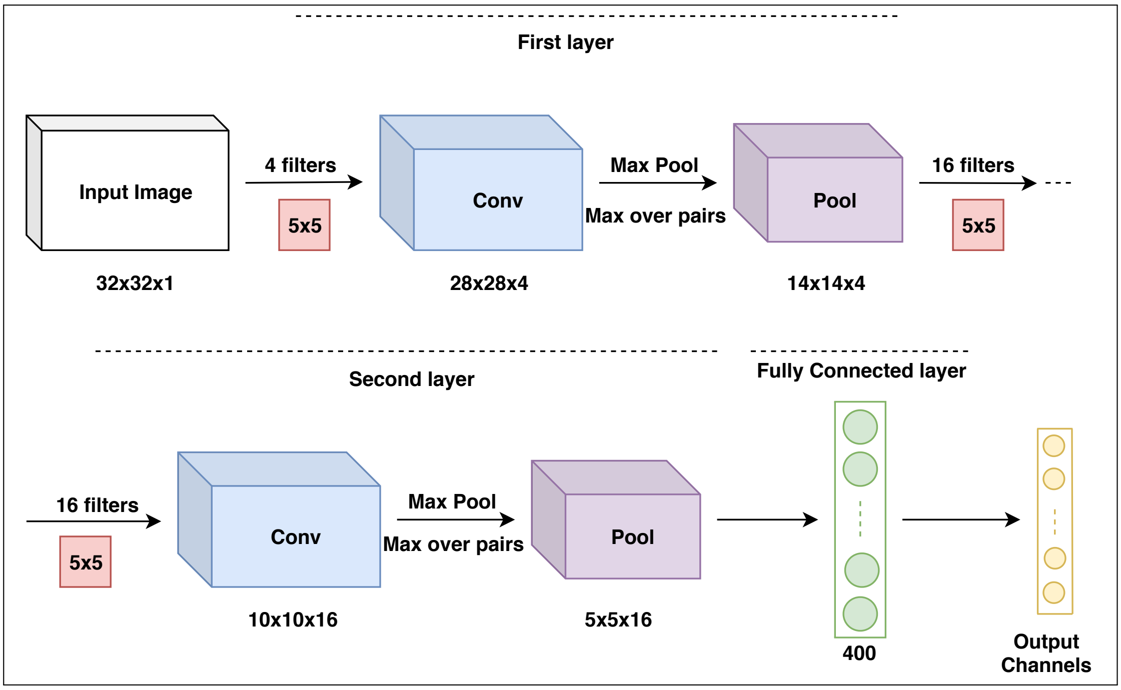


Figure 5: CNN example.

CNN is the state-of-the-art with respect to neural networks for image processing for the high performance achieved. The novel idea presented in GraphPointNet is to introduce a convolutional network suitable for more types of data, which exploits signal representation on graphs. Graphs are generic structures able to represent complex collections of data and to provide a flexible and powerful way of describing relationships between elements; they are especially useful for describing data that lie on non regular structure, like point clouds.

1.4 Graph Signal Processing

In this section a brief introduction to the graph theory is presented to understand the basic concepts and the operations involved in the project.

A generic graph $\mathcal{G}$ is constituted by a collection of vertices or nodes and edges, the connection between the nodes, respectively denoted with symbols $\mathcal{V}$ and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$.

A graph can be undirected or directed: in the first case, choosing two random vertices i and j, if the vertex i is connected to the vertex j then the vertex j is connected to vertex i; otherwise the direction of the edge is univocal, specified by an arrow, as shown in Figure 6.

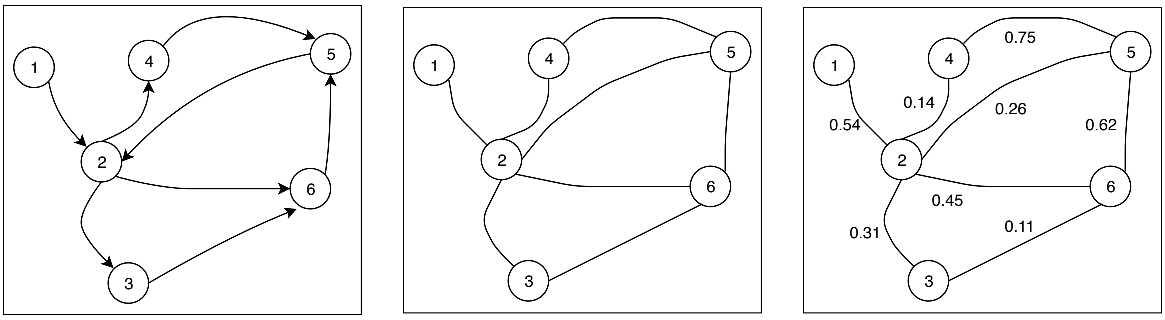


Figure 6: Graph examples. Directed Graph (top left), undirected graph (center) and weighted graph (right).

Considering a generic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_i\}$, $i \in \{1, ...N\}$, is the set of N nodes, and $\mathcal{E} = \{e_{ij}\}$, $i, j \in \{1, ...N\}$, is the collection of all the edges, the *adjacency matrix* can be defined: a matrix $\mathbf{A} \in N \times N$, where the element a_{ij} is equal to 1 if and only if the edge between the node i and the node j exists, otherwise its value is zero.

The adjacency matrix of the undirected graph in Figure 6 (center) is shown below as an example.

$$\mathbf{A}=\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

It is straightforward to understand that for an undirected graph the adjacency matrix would be symmetric.

A specific quantity, called weight, can be associated to each edge, in order to add information to the representation of the signal: in this scenario the graph is denoted as *weighted.*

Considering a weighted graph, the adjacency matrix $\mathbf{W}$ can be defined as:

$$W_{i,j} = \begin{cases} 0 & if\ e_{ij} \notin \mathcal{E} \\ w_{ij} & if\ e_{ij} \in \mathcal{E} \end{cases},$$

where w_{ij} represents the specific weight related to the edge e_{ij}.

The degree matrix related to the weighted graph in Figure 6 (right) is reported as example.

$$\mathbf{W}=\begin{bmatrix} 0 & 0.54 & 0 & 0 & 0 & 0 \\ 0.54 & 0 & 0.31 & 0.14 & 0.26 & 0.45 \\ 0 & 0.31 & 0 & 0 & 0 & 0.11 \\ 0 & 0.14 & 0 & 0 & 0.75 & 0 \\ 0 & 0.26 & 0 & 0.75 & 0 & 0.62 \\ 0 & 0.45 & 0.11 & 0 & 0.62 & 0 \end{bmatrix}.$$

After the definition of the adjacency matrix, the *degree matrix* can be introduced: it is a diagonal matrix, where the diagonal element d_{ii} is equal to the total summation of all incoming and outgoing edges relative to the node i:

$$d_{ii} = \sum_{j=1}^{N} w_{ij} \ for \ i = 1, ...N,$$

where N is the number of nodes and w_{ij} is the weight associated to the edge that connects point i and point j.

The degree matrix of the graph of Figure 6 (right) is:

$$\mathbf{D}=\begin{bmatrix} 0.54 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.70 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.42 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.89 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.63 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.18 \end{bmatrix}.$$

In the case of directed graph, edges in different directions have different signs associated to the weights.

Finally, the *Graph Laplacian matrix*, that plays a key role in graph signal processing, is presented. This matrix is defined as follows:

$$\mathbf{L} = \mathbf{D} - \mathbf{W}. \tag{1.4}$$

1.4.1 Graph Fourier Transform

Frequency domain signal analysis allows to obtain information and study several characteristics of signals; analogously it is interesting to investigate the frequency domain analysis for graph signals.

In order to perform a frequency domain analysis of a graph signal it is necessary to convert them from the graph domain to the frequency domain. One commonly exploited operator is the graph Fourier transform, whose formulation exploits the previously introduced graph Laplacian matrix. Following a brief mathematical introduction of the transform is reported.

The **L** matrix defined at the end of the previous section is symmetric and positive definite, and represents an approximation of the Laplacian operator: it is therefore possible to formulate a Fourier-like transform.

A generic graph signal $f : \mathcal{V} \rightarrow \mathbb{R}$, defined over a graph with $\mathcal{V}$ set of vertices, can be represented as a vector $\mathbf{f} = [f(1)\, f(2) ... f(N)]^T \in \mathbb{R}^N$, where N is the number of nodes in the graph and each $f(i)$ component is the function f valued at the i-th nodes. A neighborhood N_i is defined for each node i as the set of nodes connected to the node i. Given a graph signal $\mathbf{f} = [f(1)\, f(2) ... f(N)]^T \in \mathbb{R}^N$, as previously defined, the graph Laplacian operator can be seen as a difference operator [2] since:

$$(\mathbf{L}\mathbf{f})(i) = \sum_{j \in N_i} w_{ij}(f(i) - f(j)). \tag{1.5}$$

The quadratic form the graph Laplacian matrix is particularly interesting because gives information about the smoothness of the graph signal **f** according to the following formulation [3]:

$$\mathbf{f}^T \mathbf{L}\mathbf{f} = \frac{1}{2} \sum_{j \in N_i} w_{ij}(f(i) - f(j))^2. \tag{1.6}$$

The graph Laplacian matrix is real and symmetric, therefore it is possible to define orthonormal eigenvectors and eigenvalues:

$$\mathbf{L}\chi_i = \lambda_i \chi_i, \tag{1.7}$$

where $\chi_i \neq 0$ is an eigenvector of **L** and λ_i the corresponding eigenvalue.

The matrix $\mathbf{L}$ can be factorized as:

$$\mathbf{L} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T, \tag{1.8}$$

where each column of the matrix $\mathbf{Q}$ is an eigenvector of $\mathbf{L}$ and the $\boldsymbol{\Lambda}$ is the diagonal matrix whose non-zero elements are the eigenvalues of $\mathbf{L}$.

The eigen-decomposition in Equation 1.8 can be easily proved by the eigenvector-eigenvalue theory expanding the eigenvalue formulation reported in Equation 1.7.

For any continuous-time signal f_c the classic Fourier transform and its inverse are defined as:

$$\hat{f}_c(\omega) = \int \left(e^{i\omega x}\right)^* f_c(x)dx \tag{1.9}$$

$$f_c(x) = \frac{1}{2\pi}\int \hat{f}_c(\omega)e^{i\omega x}d\omega. \tag{1.10}$$

The complex exponents $\left(e^{i\omega x}\right)^*$ in Equation 1.9 are the eigenfunctions of 1-D Laplacian operator $\frac{d}{dx^2}$.

Similarly, for any graph signal $\mathbf{f} \in \mathbb{R}^N$, as previously defined, the graph Fourier transform $\mathcal{GFT}$ and its inverse $\mathcal{IGFT}$ [4] can be defined as:

$$\hat{f}(l) = \sum_{n=0}^{N-1} \chi_l^*(n)f(n), \tag{1.11}$$

$$f(n) = \sum_{l=0}^{N-1} \hat{f}(l)\chi_l(n). \tag{1.12}$$

An interesting application of the graph Fourier transform is the definition of a convolution operation over graphs in the spectrum domain.

The convolution operation in time domain is defined as follow:

$$(f_c * g_c)(x) = \int_{-\infty}^{+\infty} f_c(x-t)g_c(t)dt, \tag{1.13}$$

where f_c, g_c are continuous time signals defined in $\mathbb{R}$.

It has to be noticed that it is not possible to directly apply the classic formulation to graph signals because in the graph domain the signal translation $f(x-t)$ is not defined. Therefore, a new formulation for convolution over graph is given exploiting the graph spectral domain.

Equation 1.13 can be re-written exploiting the Fourier transform, symbol $\mathcal{F}$, and its inverse, symbol $\mathcal{F}^{-1}$, as:

$$(f_c * g_c)(x) = \mathcal{F}^{-1}\big\{\mathcal{F}\{(f_c * g_c)(x)\}\big\} = \mathcal{F}^{-1}\big\{\mathcal{F}\{f_c(x)\}\mathcal{F}\{g_c(x)\}\big\}, \tag{1.14}$$

for the convolution theorem which states that the Fourier transform of two convolved signals is equal to the multiplication of the Fourier transforms of the two signals.

The same reformulation can be applied to the graph domain; the graph-convolution operation between two graph signals f and g can be defined in the spectral domain applying Equation 1.12 and Equation 1.11:

$$(f * g)(n) = \sum_{l=0}^{N-1} \hat{f}(l)\hat{g}(l)\chi_l(n), \tag{1.15}$$

where $\hat{f}(l)$ and $\hat{g}(l)$ are respectively the graph Fourier transform of the graph signal f and g.

Therefore it is possible to define a convolution operation over a graph signal exploiting the graph Fourier transform and the spectrum domain.

1.4.2 Dynamic Edge-Conditioned Convolution

In this section the Dynamic Edge-Conditioned Convolution [1] (ECC), a generalization of the convolution operation conceived by Martin Simonovsky and Nikos Komodakis, is described and examined.

As previously discussed, the classic convolution operation can only be applied to data that lies on a regular grids. Due to the numerous applications of the classic convolution operation, it results tremendously interesting to formulate a definition of a convolution operation suitable for any type of signal, even the ones that can not be represented by regular structures or lie on non-Euclidean domains but can be easily defined on graphs, as point clouds.

In section 1.4.1 a graph convolution operation formulated in the spectrum domain is presented; however it is affected by several limitations. The most critical ones are the high computational cost and the unsuitability to deal with data with variable graph structure, due to the fact that the filters are fixed and computed in the context of the spectrum of the graph Laplacian which, to be consistent, has to be the same for all the graphs in a dataset. An alternative approach is to define the operation in the spatial domain and obtain a generalization of the classic convolution operation suitable for any type of signals, as the Edge-Conditioned-Convolution operation [1].

In [1] the convolution operation is interpreted as a local weighted aggregations over a neighborhood, involving spatial operations rather than moving to the spectral domain. One advantage of such a definition is the possibility to consider various types of graphs, with no restriction on structure and size.

We consider a generic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, constituted by n vertices and m edges. To implement the ECC a feed-forward neural network with l_{max} number of layers is exploited and the considered graph $\mathcal{G}$ has to be vertex and edge labeled: at each node and edge is associated a label according to a specific function. The function H^l associates to each vertex a feature vector at layer l, $H^l : \mathcal{V} \rightarrow \mathbb{R}^{d_l}$, where d_l is the dimension of the feature vector at layer l, and the function L associates to each edge an attribute, $L : \mathcal{E} \rightarrow \mathbb{R}^s$, where s is the dimension of the attributes.

A key concept in the algorithm is the definition of neighborhoods. A neighborhood $N^l(i)$ related to a node i at layer l is a collection of its adjacent nodes plus the node i itself:

$$N^l(i) = \{j;\ (j, i) \in \mathcal{E}\} \ \cup \ \{i\}. \tag{1.16}$$

As previously mentioned, the ECC operation computes the feature vector $H^{l+1}(i)$ related to the i vertex as a local weighted summation of signals $H^l(j)$ belonging to its neighborhood. In particular, the ECC is performed as a local operation since convolution for a node only employs feature vectors of its neighbors. Due to the locality of the operation, it is possible to manage different structures of graphs without any constraints.

A topic worthy of discussion is the definition and computation of the weight matrix $\mathbf{\Theta}_{ji}^{l}$ involved in the local weighted aggregation. We defined a filter-generating network $F^l : \mathbb{R}^s \to \mathbb{R}^{d_{l+1} \times d_l}$, that takes as input the label associated to each edge $L(j,i)$ and gives as output the matrix $\mathbf{\Theta}_{ji}^{l} = F_{\mathbf{w}^l}^{l}(L(j,i)) \in \mathbb{R}^{d_{l+1} \times d_l}$, where d_{l+1} and d_l are respectively the dimensions of feature vectors at layer $(l+1)$ and l. Notice that the function $F_{\mathbf{w}^l}^{l}$ is generated through a neural network and the term $\mathbf{w}^l$ represents the weights of such network.

In the case under study, the edge labeling function considered at each layer l is the difference between the features associated to the two nodes that an edge connects together at the layer l:

$$L(j,i) = \mathbf{h}_j^l - \mathbf{h}_i^l,\ j \in N(i)^l, \tag{1.17}$$

where the vector $\mathbf{h}_i^l$ is the feature vector of node i at l-th layer.

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

$$\begin{aligned}
\mathbf{h}_i^{l+1} &= \frac{1}{|N(i)^l|}\sigma\Big(\sum_{j \in N(i)^l} F_{\mathbf{w}^l}^{l}(L(j,i))\mathbf{h}_j^l + b^l\Big) \\
&= \frac{1}{|N(i)^l|}\sigma\Big(\sum_{j \in N(i)^l} F_{\mathbf{w}^l}^{l}(\mathbf{h}_j^l - \mathbf{h}_i^l)\mathbf{h}_j^l + b^l\Big) \\
&= \frac{1}{|N(i)^l|}\sigma\Big(\sum_{j \in N(i)^l} \mathbf{\Theta}_{ji}^{l}\mathbf{h}_j^l + b^l\Big),
\end{aligned} \tag{1.18}$$

The symbol σ represents the activation function selected for the neural network.

CHAPTER 2

STATE OF THE ART ON POINT CLOUD PROCESSING

Different typologies of point cloud processing that address various tasks such as classification, segmentation and denoising [5] [6] [7] [8] [9] are available in literature, revealing a growing interest in point clouds and their applications.

In this section particular attention is given to point cloud denoising: the importance of this task is discussed and the state-of-the-art methods are presented, highlighting the differences and common points among them.

2.1 Point Cloud Denoising

All the available point cloud acquisition methods, such as radars, lasers, sensors or even CAD tools able to create point clouds from images, insert non-negligible noise, corrupting the collected information and requiring a denoising pre-processing before any further use. Therefore, the task of point cloud denoising has become a problem investigated by several researchers developing different techniques.

Traditional approaches can be mainly divided into two categories: local and non-local operator based. Local operator based methods generally implement surface fitting of noisy inputs; the Moving Least Squares (MLS) is one of the most common, it projects each noisy input point to a surface approximated from the noisy observation. Other widely used approaches are based on sparse representations of geometric characteristics, such as normals to the surface, estimated solving minimization problems. On the other hand, the methods belonging to the second category exploit self similarities to re-construct noiseless point clouds; one of its most promising branch is a graph-based approach, which exploits graph representation of point clouds and graph properties. In particular, methods that involve signal smoothness and regularization such as graph Laplacian regularization or total variation are particularly efficient.

Recently new techniques involving neural networks have been investigated. A few networks have been presented and the most relevant are PointCleanNet [6] and Neural Projection Denoising (NPD) [7], a network based on local surface estimation. Both networks are based on the architecture of PointNet[5], the first neural network designed to deal with point clouds for classification and segmentation tasks.

None of the neural networks proposed for the denoising of point clouds implements a convolutional network, it is coarsely emulated using multilayer layer perceptrons and max-pooling layers.

2.2 Notable Methods for Point Cloud Processing

In the following a brief review of representative and notable projects concerning the point cloud processing is reported.

2.2.1 PointNet

PointNet [5] is the first neural network able to efficiently deal with point clouds for classification and segmentation tasks. The architecture, shown in Figure 7, is quite simple, characterized by several multilayer perceptrons, MLP, and maxpooling layers, MP.

It can be seen in Figure 7 that two transformer blocks STN_1 and STN_2 are inserted, both have an internal architecture similar to the global structure of PointNet: several fully connected layers with a final maxpooling layer. The STN_1 block is an input transformer that creates a 3×3 matrix from 3-D input point cloud in order to constrain the network to be rotation invariant. The second block, STN_2, is a feature transformer similar to the previous one but returns as output a 64×64 matrix. Each transformer consists in shared multilayer perceptrons with output dimensions (64,128,1024), a maxpooling and two fully connected layers to regress the dimensions.

The number of input points and the size of the maxpooling layer's output have a great impact on the performances of the net, hence it is reasonable utilize only point clouds with a limited number of points.

PointNet is a neural network designed for point clouds, characterized by a simple architecture and able to achieve promising results for the addressed tasks. Several networks, which exploit similar architectures but address different processing tasks, have been realized; the most notables are PCPNet [8] for estimation of local shape properties such as normals or curvature of the points, PointCleanNet [6], for point cloud denoising, which is inspired by PCPNet, and Neural Projection Denoising [7], NPD, based on local surface estimation.

In the following PointCleanNet and NPD are well discussed; instead an in-depth review of PCPNet is not presented because the architecture would not introduce any novel aspects and the task is beyond the topic of the thesis.

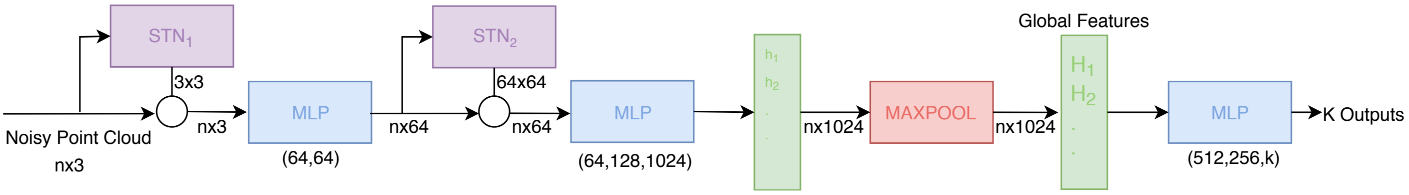


Figure 7: PointNet: architecture for point cloud classification.

2.2.2 PointCleanNet

PointCleanNet [6] is a deep neural network able to remove outliers and denoise the remaining points. Outliers are defined as additional points outside of the original surface of the point cloud and they can be removed in order to increase the quality of the data. Given as input a noisy point cloud, the network is first trained to learn surface patches to identify the outliers and then to estimate the displacements of the remaining points affected by white additive noise.

The training set is a collection of patches created by selecting a point and its closest points within a specific radius; the algorithm only performs the denoising of the center point of the patch, which depends only on a local neighborhood, treating the denoising task as a local problem.

The network is divided into two stages. The first stage is dedicated to the outlier removal, where an outlier probability o_i associated to each point is predicted and if it is larger than a threshold the point is considered an outlier. After this procedure the identified outliers are removed from the input and then the second stage is performed. The displacements d_i of the remaining points to the original and unknown surface are estimated and applied to the noisy point cloud without outliers in order to obtain the denoised point cloud.

The structure exploited in both of the steps can be seen in Figure 8 and Figure 9. The global architecture is largely drawn from PCPNet [8], which is inspired from PointNet. The goals of the network are to move the noisy points as close as possible to the original surface and maintain a regular distribution avoiding clustering. The total loss function in Equation 2.2 exploits the distance between each denoised points and its closer point in the ground truth point cloud to move the point closer to the surface, L_s term in Equation 2.1, and a regularization term to avoid clustering, L_r in Equation 2.1.

$$L_s(\tilde{\mathbf{p}}_\mathbf{i}, \mathbb{P}_{\tilde{\mathbf{p}}_\mathbf{i}}) = \min_{\mathbf{p}_\mathbf{j} \in \mathbb{P}_{\tilde{\mathbf{p}}_\mathbf{i}}} ||\tilde{\mathbf{p}}_\mathbf{i} - \mathbf{p}_\mathbf{j}||_2^2 \, , \qquad L_r(\tilde{\mathbf{p}}_\mathbf{i}, \mathbb{P}_{\tilde{\mathbf{p}}_\mathbf{i}}) = \max_{\mathbf{p}_\mathbf{j} \in \mathbb{P}_{\tilde{\mathbf{p}}_\mathbf{i}}} ||\tilde{\mathbf{p}}_\mathbf{i} - \mathbf{p}_\mathbf{j}||_2^2 \, , \tag{2.1}$$

$$L_a = \alpha L_s + (1 - \alpha) L_r. \tag{2.2}$$

In Equation 2.1 $\tilde{\mathbf{p}}_\mathbf{i}$ is the denoised point i, $\mathbf{p}_\mathbf{i}$ is the original point i and $\mathbb{P}_{\tilde{\mathbf{p}}_\mathbf{i}}$ is the neighborhood of points closer to the denoised point i in the original point cloud. In Equation 2.2 α is a parameter to weight the importance of the regularization term in the total loss.

In Figure 8 and Figure 9 the architecture of the networks proposed in [6] is shown, where spatial transformer layers, *QSTN* and *STN* blocks, and fully connected layers, *FNN*, are exploited. Both the networks, for outlier removal or denoising, share the same architecture. The first layer, *QSTN* is used to constrain the network to be rotation invariant, notice that the same block is applied again at the end of network to restore the original rotation and make it consistent with the input data. The $QSTN$ block has the same function of the first transformer block in PointNet, but differs for the output dimension that in this case is a quaternion. The subsequent layers perform the feature extraction, which is realized with several fully connected layers and a full linear transformation, the STN block. The features of each points, h_i, are combined together with a symmetric operation, a maxpool or a summation, to obtain an order invariant feature vector, H_i. At last, a regressor, composed of fully connected layers, estimates the desired output: the displacements d_i or the outlier probability o_i.

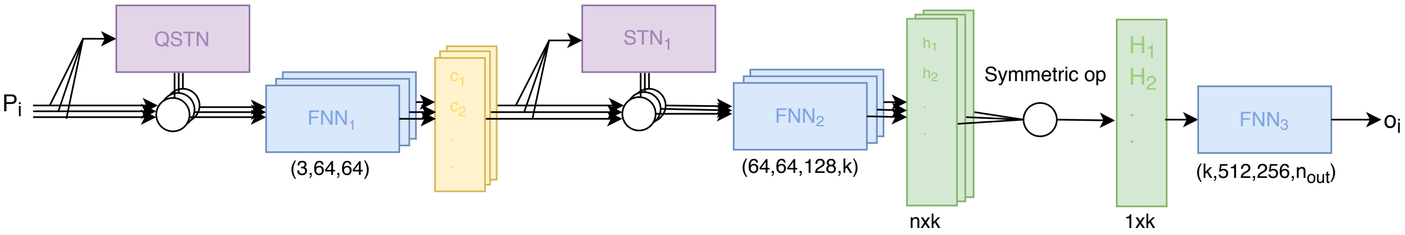


Figure 8: PointCleanNet: architecture of the outlier removal step.

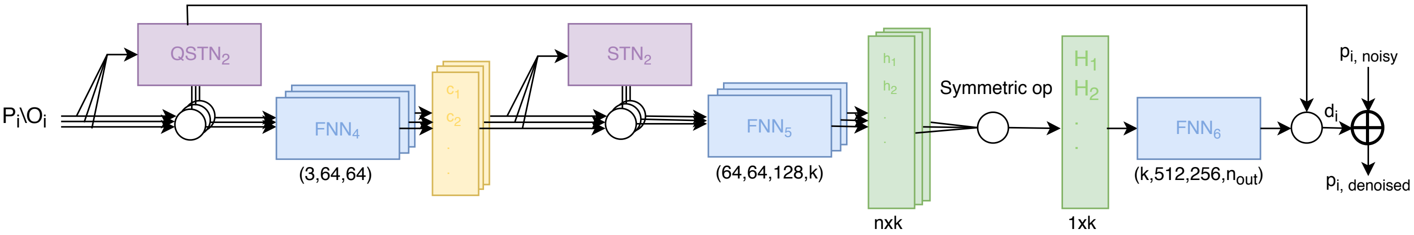


Figure 9: PointCleanNet: architecture of the denoising step.

PointCleanNet is able to outperform currently state-of-the-art traditional methods, exploiting the simple architecture of PointNet re-arranged for the denoising task. Furthermore, it is a blind network: the network is not trained for a specific amount of noise, but the same network can denoise point clouds affected by white additive noise with different standard deviations. However, the network is able to denoise just the center point of the patch by time, performing a point training rather than a patch training.

2.2.3 3-D Point Cloud Denoising via Deep Neural Network Based Local Surface Estimation

The Neural Projection Denoising [7], *NPD*, is a novel network which exploits the estimation of geometrical properties to denoise a corrupted point cloud.

The network predicts reference planes from a noisy input and projects the points to the surfaces. The network has to learn from local and global features the reference planes $\hat{T}_i$

characterized by normal vector $\hat{\mathbf{a}}_i$ and intercept $\hat{c}_i$ for each noisy point $\tilde{\mathbf{p}}_i$ and then obtain the denoised point $\hat{\mathbf{p}}_i$ performing a projection:

$$\hat{\mathbf{p}}_i = \tilde{\mathbf{p}}_i - \hat{\mathbf{a}}_i^T \tilde{\mathbf{p}}_i \hat{\mathbf{a}}_i + \hat{c}_i \hat{\mathbf{a}}_i. \tag{2.3}$$

During a pre-processing phase, the true reference planes are computed from the original point cloud with graph-based methods. The ground truth reference planes jointly with the noiseless point cloud are used to supervise respectively the estimation of the surfaces and the final denoised version of the data. To evaluate the quality of the final denoised point clouds with respect to the original one the mean squared error is computed:

$$MSE = \frac{1}{N} \sum_{i=0}^{N} ||\hat{\mathbf{p}}_\mathbf{i} - \mathbf{p}_\mathbf{i}||_2^2, \tag{2.4}$$

where $\hat{\mathbf{p}}_i$ is the estimated denoised point i, $\mathbf{p}_i$ is the original point i and N is the number of points.

Instead, the cosine similarity is exploited as loss function to constrain the reference plane estimation:

$$\cos(\{\hat{\mathbf{a}}_i, \hat{c}_i\}, \{\mathbf{a}_i, c_i\}) = \frac{1}{N} \sum_{i}^{N} \left| 1 - \frac{[\mathbf{a}_i^T, c_i]^T [\hat{\mathbf{a}}_i^T, \hat{c}_i]}{||[\mathbf{a}_i^T, c_i]||_2 ||[\hat{\mathbf{a}}_i^T, \hat{c}_i]||_2} \right|, \tag{2.5}$$

where N is the number of points, $\hat{\mathbf{a}}_i$ is the estimated normal vector of point i, $\hat{c}_i$ is the estimated intercept of point i and $\mathbf{a}_i$ and c_i the respectively ground truths. The total loss is the combination of Equation 2.4 and Equation 2.5.

The architecture is shown in Figure 10. It can be seen that the transformer blocks, typical of the PointNet architecture, are not inserted. A *MLP* is exploited to extrapolate local features and a maxpooling operation to extract global information. The concatenation of the two features such predicted is the input of a series of *MLP* that estimates the reference planes. Finally, the noisy points are projected on the estimated reference planes, obtaining the denoised point cloud. The strength of the method is the simplicity of the architecture: it directly estimates surfaces and project the points.

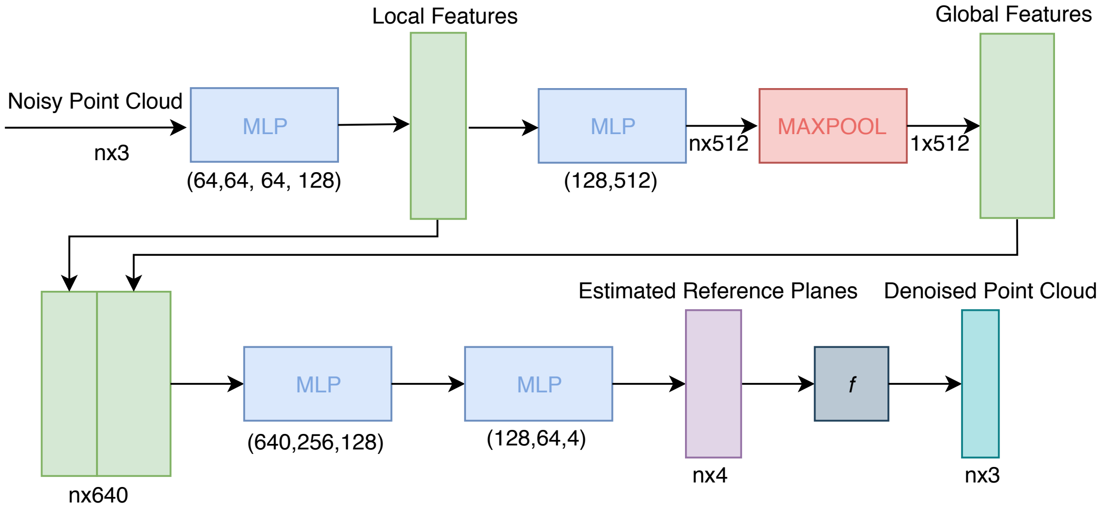


Figure 10: Neural Projection Denoising: architecture of the whole network.

CHAPTER 3

PROPOSED APPROACH FOR POINT CLOUD DENOISING

In this section different aspects of the first stages of network's design are described: the general structure and the creation of the datasets.

3.1 General Structure

The method proposes a deep neural network to denoise point clouds based on graph-convolutional layers, the first step of the design is to outline a general structure of the network.

In the context of image denoising, it has been recently demonstrated [10] that convolutional neural networks achieve better results if trained to estimate the residual, the difference between the noisy and the clean image, rather than directly the denoised image. Residual learning of convolutional neural network, introduced in [11], was originally conceived to address the performance degradation with increasing network depth phenomenon. Also the proposed architecture adopts the residual learning principles: the network is trained to predict the residual between the noisy and the clean point cloud, i.e. the additive white noise.

The core of the architecture is composed of several blocks, called *Residual Block*, drawn by [11], able to detect and remove the correlations of the input of the block itself: the ground-truth point cloud is gradually removed from the noisy observation, ideally obtaining only white additive noise at the end of the net. The estimated noise is subtracted to the noisy input in order to finally obtain the denoised point cloud. An in-depth discussion of the structure is reported in section 4.2.3.

The proposed network is based on graph-convolutional layers, which are the basic components of each residual block. All the operations of these blocks are defined in the feature space rather than in the 3-D space, then a *Pre-Processing Block* is introduced at the beginning of the network to project the noisy point cloud from the 3-D space to the feature space, extrapolating meaningful information.

In Figure 11 it is possible to observe the general architecture described: only two residual blocks are reported as an example, the actual number is discussed in section 4.2.3, where a wider explanation of each block is reported.

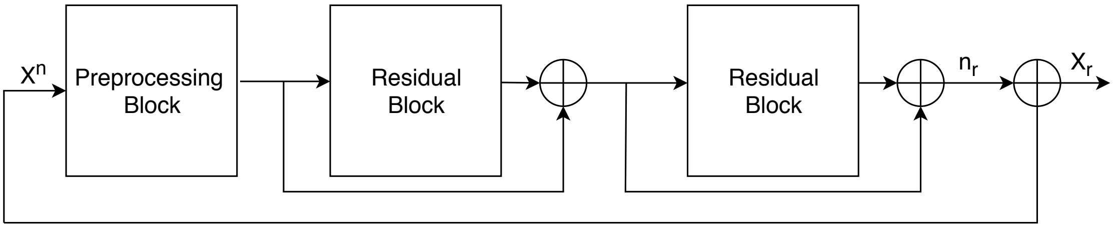


Figure 11: Block diagram of the general structure proposed.

3.2 Dataset creation

An important aspect worthy of discussion is the creation of the dataset: the ModelNet40 [12] database is exploited for the training and validation set; instead the Shapenet [13] repository for the testing set.

Both the previous cited archives store 3-D representations of objects belonging to different common categories: airplane, bath, bed, car, chair and so on.

ModelNet40 is provided by the University of Princeton and it is constituted by a collection of 3-D CAD models of objects divided into 40 different categories, all cleaned by the authors. The data are stored in .OFF format file and each point cloud is represented by a collection of meshes, which are geometrical portions of the surface. In each file the spatial coordinates of the *vertices* and the *faces*, expressed as indices of vertices that compose them, are reported. On the other hand, Shapenet, released by the University of Stanford, is a large-scale dataset of 3-D data, including 55 common object categories with more than 50.000 models in .obj format file, where the data are reported as in the other cited dataset.

All the data that constitute the training, the validation and the testing set have to be pre-processed in order to apply the denoising method to point clouds with the same initial conditions. As first step a mesh-sampling on the faces is performed, obtaining a collection of points in spatial coordinates: for the training and validation data 20.000 points per point cloud are sampled, instead for the testing data 30.720. The points are sampled following a normal distribution to avoid clustering.

After the sampling, all the point clouds are normalized: each point cloud is scaled in order to be contained in a sphere with unitary diameter. The diameter is defined as the measure of the maximum distance between two points in the point cloud. This operation is necessary because point clouds can have an arbitrary diameter that would not affect the representation itself of the data, but it has to be taken into account when an addictive white noise is applied: noise with same value of standard deviation would have different effects if applied to point clouds with different scales and diameters. A further explanation about the application of the noise is reported in section 3.3.

The last step of the data pre-processing consists in the patch division of the data only for the training and validation set. As already explained, the graph-convolutions perform operations upon points and their neighborhoods, therefore the patch has to be created accordingly. For the training and the validation set several points and their 1023 closest points are selected from all the sampled and normalized point clouds, the central points are chosen sufficiently spaced to obtain non-overlapped patches; each center point and its 1023 closest points constitute a single patch. The final datasets for the training and validation phase are respectively composed of 117.000 and 100 different patches extracted from ModelNet. Instead the testing set is composed of 100 point clouds extracted from Shapenet, belonging to ten categories not included in the training or validation set. For the testing set the patch division is not implemented.

3.3 White Noise Application

Point clouds from the datasets described in section 3.2 are artificially corrupted in order to model a noisy data observation. In particular, white additive noise is considered in this project;

it affects point clouds changing the position of each points, modifying the value of the three spatial coordinates.

An easy way to emulate this behaviour is to create a matrix of the same dimension of the clean input filled with normally distributed values with zero mean and σ as standard deviation and add the obtained matrix to the noise-free input, the outcome would be a noisy version of the clean input:

$$\mathbf{X}^n = \mathbf{X} + \mathbf{n}, \tag{3.1}$$

$$\mathbf{n} = \mathcal{N}(0, \sigma^2), \tag{3.2}$$

where $\mathbf{X}^n$ is the noisy point cloud, $\mathbf{X}$ is the original clean input, $\mathbf{n}$ is the white additive noise and $\mathcal{N}(0, \sigma^2)$ is a normal distribution with standard deviation σ.

It is clear that in order to directly apply the chosen value of standard deviation and have consistent noisy point clouds, all the original input has to be normalized with a diameter equal to one, otherwise applying same standard deviation would have different effects to point clouds with different diameter.

The proposed neural network is able to learn how to reconstruct data affected by a specific amount of noise, therefore the training and the testing have to be performed exploiting data characterized by the same standard deviation in order to obtain consistent results.

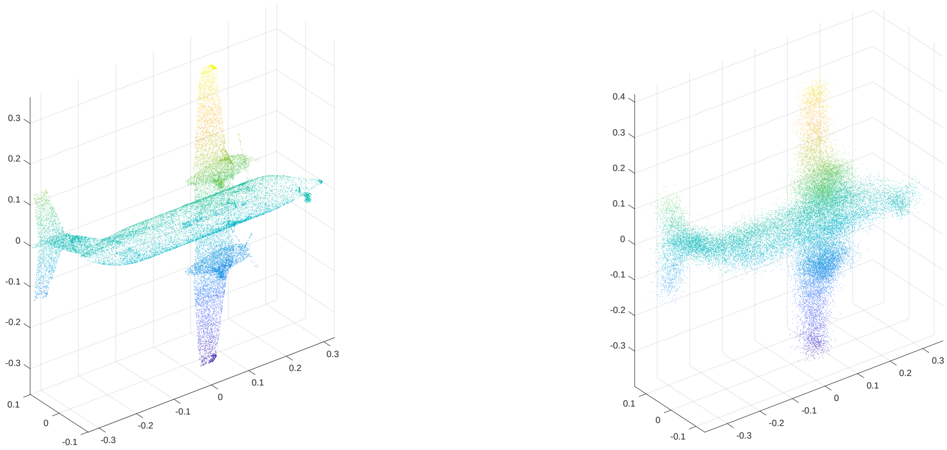


Figure 12: Examples of corrupted point cloud. Ground truth (left) and noisy corrupted by white noise with σ=0.02 (right).

CHAPTER 4

GRAPH NEURAL NETWORK FOR POINT CLOUD DENOISING

In this section an in-depth analysis of the network proposed and the details of the main blocks are reported.

4.1 Architecture design

4.1.1 Overview

An overview of the proposed architecture, called GraphPointNet, is shown in Figure 13. It can be seen from the figure that the network is divided into three big blocks: the pre-processing block, two residual blocks and the final block where the reconstruction of the denoised point cloud is performed.

The pre-processing block is designed to project the noisy observation from the 3-D space to the feature space. During the training, the network progressively learns an efficient feature description, able to detect meaningful information for the required task.

After, the feature expansion, the data are processed by several residual blocks. The network exploits the residual learning and predicts the white additive noise instead of directly the denoised point cloud. Each block detects and removes correlations in the input of the block itself to provide as output a progressively better estimation of the white additive noise; the idea is to gradually eliminate the true point cloud and obtain only white noise.

Finally, in the last step, the estimated white additive noise in the feature space is projected back to the 3-D domain to be consistent with the noisy input, then it is subtracted to the noisy point cloud and the denoised point cloud is obtained.

4.1.2 Design choice

The core of the architecture is composed of several graph-convolutional layers, followed by batch normalization blocks and Leaky Relu as activation functions as shown in Figure 13.

The Leaky Relu function is a variant of the Relu function described in section 1.4.2: for positive values it is equal to the Relu function and for negative ones it returns small negative rate instead of being equal to zero. The Leaky Relu function is used to overcomes the *"dying Relu"* problem: in a network with only Relu functions, if a neuron receives a negative input it would return as output always zero, as shown in equation Equation 1.3, in the case in which many neuron's input become negative for any reason, random initialization or wrong learning rate, they would never work.

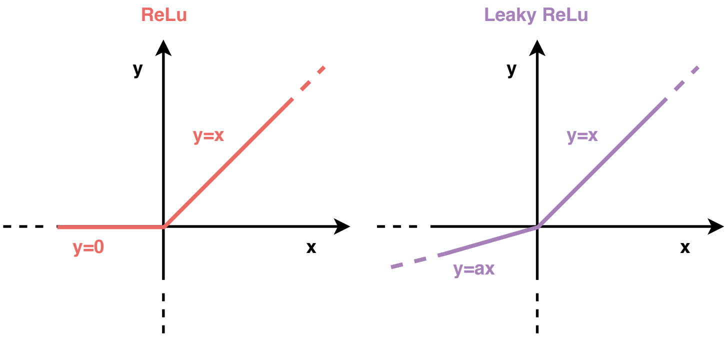


Figure 14: Relu vs. LeakyRelu.

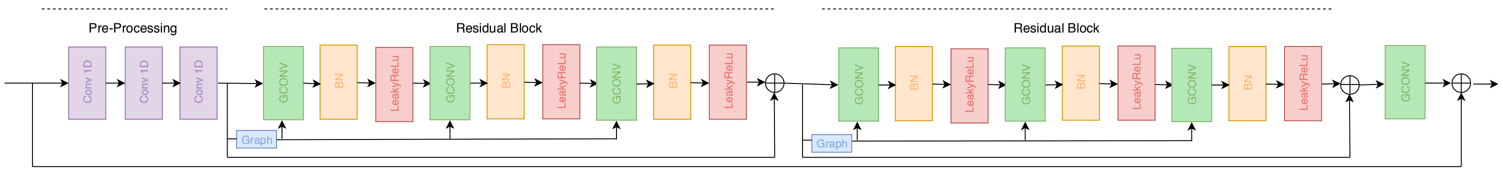


Figure 13: GraphPointNet: architecture of the network proposed.

A noisy point cloud can be represented as:

$$\mathbf{X}^n = \mathbf{X} + \mathbf{n}, \tag{4.1}$$

where $\mathbf{X}^n$ is the noisy point cloud, $\mathbf{X}$ is the clean point cloud and $\mathbf{n}$ is the additive white noise, all matrices have the same dimension $N \times 3$, N number of points in the point cloud.

As already mentioned, the network learns to predict the white noise $\mathbf{n}$, that represents the discrepancy between the noisy observation and the clean point cloud. This structural choice is based on the work of Zhang et al. [10], where it has been shown that a residual network is very efficient for image denoising tasks. Zhang et al. propose an architecture for image denoising characterized by a single residual unit: a sequence of convolutional layers, batch normalizations and Relu functions. It can be noticed that the same structure can be exploited to train either the residual mapping or directly the denoised image, it depends only on the loss function. The advantages of the residual learning are exposed in [11], where the phenomenon of performance degradation when increasing the depth of the net is analyzed. Contrary to what might be expected, adding layers to an optimized network leads to increment of the training error, solvers are not able to find a solution as good as the one associated to the original structure even if the deeper version of an architecture is created only with identity mapping as additional layers. A deep residual network is proposed [11] to overcome this degradation, the net instead of predicting the desired mapping $H(x)$, estimates the residual $R(x) = H(x) - x$, where x is the input of the net. An inverse operation is applied to finally recover the original desired function $H(x)$: $H(x) = R(x)+x$. An example of building block is reported in Figure 15.

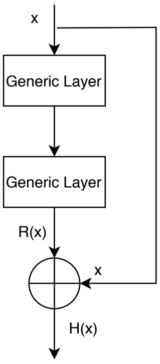


Figure 15: ResNet: building block of residual learning.

It is possible to assume that a network is not able to well approximate an identity function based on the fact that even networks with inserted identical mapping present degradation problem and therefore the network reformulation with residual learning is justified and useful. Exploiting a residual learning can bring advantages if the desired function is close to the identity: the network learns the discrepancies between the reference and the identity rather than a new function. The residual formulation $H(x) = R(x)+x$ is realized just inserting a short connection that turns the original design into its residual version without increasing the complexity or adding parameters.

In the proposed denoising network several residual connections are inserted; as shown in Figure 13, two internal residual blocks are designed besides the connection between the noisy input and the estimated noise to recover the denoised point cloud.

The main novel contribution of the method proposed is the insertion of graph-convolutional layers, that allows to design a CNN-like structure also for input that relies on a non regular structure such point clouds. Each graph-convolutional layer implements the Edge-Conditioned-Convolution defined as:

$$\begin{aligned}\mathbf{h}_i^{l+1} &= \frac{1}{|N(i)^l|}\sigma\Bigg(\sum_{j\in N(i)^l} F^l_{\mathbf{w}^l}(\mathbf{h}_j^l - \mathbf{h}_i^l)\mathbf{h}_j^l + b^l\Bigg)\\ &= \frac{1}{|N(i)^l|}\sigma\Bigg(\sum_{j\in N(i)^l} \mathbf{\Theta}^l_{ji}\mathbf{h}_j^l + b^l\Bigg),\end{aligned}$$

where the vector $\mathbf{h}_i^{l+1}$ is the feature vector of the node i at the $(l+1)$-th layer, $\mathbf{h}_i^l$ is the feature vector of node i at l-th layer, $N(i)^l$ is the neighborhood of the node i at l-th layer, σ is the activation function and b is a bias. The term $F^l_{\mathbf{w}^l}(\mathbf{h}_j^l - \mathbf{h}_i^l)$ represents a function computed through a deep neural network with training parameters $\mathbf{w}^l$ at layer l that takes as input the difference between the feature vector at layer l of the center point i and its neighbor j and returns a weight matrix $\mathbf{\Theta}^l_{ji}$. The matrix $\mathbf{\Theta}^l_{ji}$ stores the weights associated to each point j in the neighborhood of the point i. The function $F_{\mathbf{w}^l}$ is originally designed in [1] as a deep neural network of two fully connected layers, but this implementation negatively affects the training with over-parameterization. In order to overcome this problem, a partially structured matrix is exploited in the last layer of the network that implements the $F_{\mathbf{w}^l}$ function: multiple row-subsampled circulant matrices are stacked, decreasing the number of parameters. Partial circulant matrices with three rows for each has been chosen, constraining the network to have a feature number multiple of three.

The proposed network is finalized to denoise point clouds, therefore the *Mean Squared Error* between the denoised point cloud $\hat{\mathbf{x}}$ and its ground truth $\mathbf{x}$ is exploited as loss function:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} ||\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i}||^2, \tag{4.2}$$

where N is the number of points in the point cloud. The loss function is computed after the estimated noise is projected in the 3-D space and subtracted to the noisy point cloud.

4.2 Block Diagram

A brief focus on the most characteristic building blocks is reported. In particular the functionality and the implementation of the blocks are discussed.

4.2.1 Pre-Processing Block

The pre-processing block is in charge of the feature expansion of the noisy observation: in this specific section, the network should extract the important information from the input data that will be used for the denoising task.

Intuitively, the more features are involved in the features domain, the more information the network is able to detect. On the other hand, enlarging the number of feature causes a non negligible increment of occupied memory and after a specific performance the network is not able to detect more information, even after further expansions. It has to be highlighted that the computation of the graph convolution, the main operation of the network, requires a large amount of memory, therefore it is necessary to find a trade-off between the number of features and the memory occupancy. In early simulations the number of features chosen is 66, successively enlarged to 120.

Several 1-D convolutions are exploited for the implementation of the feature extraction. Each of these operations can be seen as a multiplication by a matrix plus a scalar quantity, called bias. In Figure 16 it is possible to observe the implementation with 66 features: the number of features is progressively increased from 22 to 44 and finally to 66. Gradually enlarging the features number in a deep structure achieves better results and a more stable representation than performing the expansion directly with one layer. The batch normalization layers are inserted in order to obtain consistent data with the rest of the network.

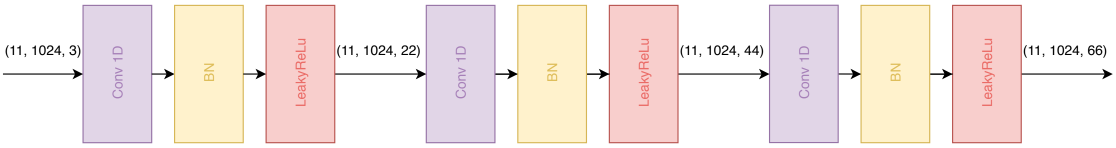


Figure 16: Architecture of pre-processing block.

4.2.2 Residual Block

The purpose of the residual blocks has been largely discussed in the previous chapters, here just the design description is reported. The simulations are performed with two residual blocks, due to the high-time consuming operations involved; adopting a deeper model would only have slowed down the process, especially in the first stages of the development.

Each residual block is composed of three layers of graph-convolution which takes two inputs: the feature vectors of all the points and a graph, which is computed at the beginning of each block. A graph is a matrix that stores the difference between all the feature vectors. Notice that the graph construction is dynamic, i.e. the graph is updated after every residual block but shared among the graph convolutional layers inside a block to limit computational complexity. Each graph-convolutional layer in a residual block is followed by a batch-normalization layer and an activation function, as visible in Figure 17. The input data are normally distributed, an important characteristic that helps the network in the training phase, but as the network gets deeper this property is easily lost, the batch normalization block is in charge of the normalization of the data through the net. The activation function exploited in the whole project is the Leaky-ReLu function, as reported in the Figure 17.

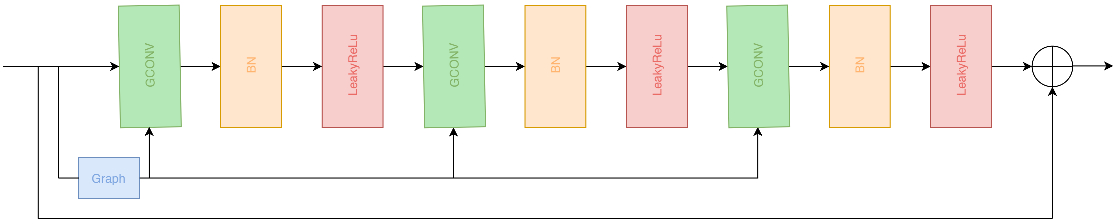


Figure 17: Architecture of a residual block.

4.2.3 Graph Convolutional layer

The *Graph Convolutional layer* is the core of the network: it takes as input the feature vector matrix at layer l and a graph and returns for each point i the feature vector $\mathbf{h}_i^{l+1}$ that becomes the input of the following layer $(l+1)$. The output is computed by performing a weighted local aggregation over a neighborhood identified by the graph:

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

where $N(i)^l$ is the neighborhood of the point i at layer l, $\mathbf{h}_j^l$ is the feature vectors associated to the node j at layer l, $\mathbf{\Theta}_{ji}^l$ is the weight matrix at layer l of node j in $N(i)^l$, σ is the activation function and b is a bias.

The term "local" used in the description of the operation means local in the feature space. The peculiarity of the graph convolution operation is to consider close into the feature space not the points that are spatially nearby, but points that share similarities according to some metric.

In order to detect the points belonging to the neighborhood of each point, it is necessary to build the edge-labeled graph of the point cloud. In our network the label of each edge is the difference between the feature vector associated to the nodes that the edge connect together. First of all, the distances between all the points are computed, then the neighborhood of each node can be defined: for each node i a fixed number of closest points is selected, building a k-nearest neighbors graph. Different numbers of neighbors are tested: early simulations employ eight points and then the value is increased, reflecting an increment of the network global performance. After the definition of the neighborhood the weighted aggregation is performed.

In Figure 18 the implementation of the graph-convolution operation is reported. The block takes as input the feature vector matrix $\mathbf{H}^l$, which contains the feature vectors at layer l associated to all the points, and the graph, implemented by the block called *Graph* outside the main element *Gconv* in Figure 18. In the *Graph* block the distances between all the points in terms of Square Euclidean Distance are computed. Inside the *Gconv* block a sub-block called *Non-local/local aggregation* is reported, where the weighted aggregation is implemented for each point j in the neighborhood of the center point i, except for the point i itself. Our experiments, reported in chapter 5.2.1, consider eight neighbors for each point. An additional block, called *1-D convolution* block, is exploited to take into account the contribution of self loops, i.e. the center point i of each neighborhood, in the local weighted aggregation. The mean of the nine components, eight neighbors and one self-loop, described above is considered to obtain the output feature vector matrix $\mathbf{H}^{l+1}$.

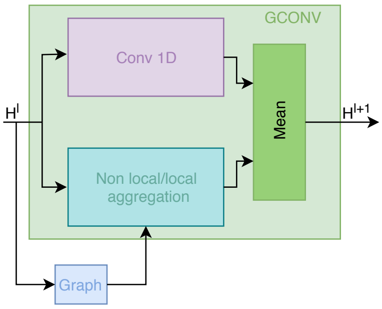


Figure 18: Architecture of graph convolution layer: H^l is the input of the layer l and H^{l+1} is the output of the graph convolution layer that will be the input of the $(l+1)$ layer.

CHAPTER 5

VALIDATION AND RESULTS

Once the network is trained, several tests are performed in order to evaluate the performances, the quality of the results and make comparisons with other methods.

The dataset of the training phase is composed of non-overlapped patches extrapolated from a point cloud. A patch is created selecting a point, that will be the center of the patch, and its 1023 nearest neighbors points, a detailed explanation is in section 3.2. During the testing phase the convolutional structure of the training data has to be replicated properly, but the point cloud has to be denoised without points overlapping.

For each point cloud in the testset the nearest 32 neighbors are individuated and saved in a matrix, called *nearest-neighbors matrix*, that the network takes as input with all the points of the point cloud. A specific code for the testing net is edited to perform properly the convolution operation over the whole point cloud. All the non-convolutional operations remain unchanged from the description in the training code because they can be directly computed over the whole point cloud. Instead, the convolutional operations are executed just over one point at a time extracting its nearest neighbors from the nearest-neighbors matrix computed a priori: a convolution operation takes as input the feature vectors associated to one center point and to the points in its neighborhood and returns as output just the feature vector of the center point. This operation is executed in parallel and the whole point cloud after a graph-convolutional layer is collected and becomes the input for the following layer.

Once the denoised point clouds are obtained, the *Chamfer measure*, also called *Cloud-to-Cloud distance* (*C2C*), is exploited to evaluate the quality of the method. This metric is adopted in several papers concerning the point cloud denoising, such as [6] and [9], but occasionally called in different ways despite the same computation. It consists in measuring the mean distance between the points of the denoised and the clean point cloud. More in detail, it computes the mean of the square root of the squared Euclidean distances between each point of the ground truth and its closest denoised point, then computes the opposite mean distance, between each denoised point and its closer point in the ground truth point cloud; the final C2C distance is the mean between the two distance estimations. In the following the equation is reported:

$$C2C = \frac{1}{2}\Big[\frac{1}{N_1}\sum_{\mathbf{x}_i \in N_1} \sqrt{\min_{\hat{\mathbf{x}}_j \in N_2} ||\mathbf{x}_i - \hat{\mathbf{x}}_j||_2^2} + \frac{1}{N_2}\sum_{\hat{\mathbf{x}}_i \in N_2} \sqrt{\min_{\mathbf{x}_j \in N_1} ||\hat{\mathbf{x}}_i - \mathbf{x}_j||_2^2}\Big], \tag{5.1}$$

where N_1 and N_2 are respectively the points of the clean and of the denoised point cloud, $\mathbf{x}$ is the clean point cloud and $\hat{\mathbf{x}}$ is the denoised point cloud.

The testset is composed by ten different point clouds from ten categories of the dataset Shapenet [13]; the data are pre-processed as discussed in section 3.2. In the tables below the results of several simulations are reported, exploiting different feature numbers and applying several levels of noise.

For each simulation one table for each category is reported, where the results for each of the ten models are presented; the Shapenet [13] dataset contains a large amount of point clouds for each category, therefore the number of each point cloud is shown in the left column.

The simulations differ for number of features and standard deviation, the presented tests are performed with the following characteristics:

- Network with 66 number of features and 0.02 of standard deviation
- Network with 66 number of features and 0.01 of standard deviation
- Network with 120 number of features and 0.02 of standard deviation

The Graph Laplacian Regularization [9] method and PointCleanNet [6] are taken into account as state-of-the-art methods for point cloud denoising. The authors of the first project (*GLR*) kindly provided the MATLAB code that is executed over the same testing set of GraphPointNet, in order to have comparable results. The code of PointCleanNet and a pre-trained model for a blind denoising are available on github. The network is tested over the same testset of our experiments.

Following the results in terms of C2C distances are reported and discussed.

5.1 Simulations with noisy point clouds at high level of noise

5.1.1 Quantitative Results

The first test performed considers GraphPointNet trained for 900.000 iterations, with feature number equal to 66. The testset is the same for all three methods and it is affected by additive white noise with standard deviation 0.02.

TABLE I: AIRPLANE TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.02$.

Airplane				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_000492	0.009181	0.007496	0.005488	**0.004984**
model_003733	0.009728	0.008604	0.007418	**0.003890**
model_006263	0.009083	0.007373	0.006298	**0.005559**
model_022125	0.009274	0.007216	0.007489	**0.007102**
model_022283	0.009245	0.007257	0.005614	**0.005459**
model_023833	0.009129	0.007313	0.005672	**0.005188**
model_026886	0.009108	0.007240	**0.005680**	0.005946
model_031422	0.009492	0.007565	0.006623	**0.006224**
model_034021	0.009658	0.007728	0.006093	**0.005029**
model_044620	0.008440	0.006912	0.006156	**0.005521**

TABLE II: BENCH TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.02$.

Bench				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_005965	0.009481	**0.007239**	0.007655	0.007404
model_016245	0.001003	0.008126	0.006851	**0.004236**
model_022257	0.009937	0.008197	0.006523	**0.005727**
model_033008	0.007714	0.006276	0.005726	**0.005402**
model_033970	0.009340	0.007057	0.005943	**0.004966**
model_035602	0.009721	0.008139	0.006334	**0.006332**
model_040561	0.008675	0.006786	0.006535	**0.005313**
model_040935	0.009151	0.007274	0.006271	**0.004408**
model_048967	0.009819	0.007458	0.00671	**0.005335**
model_050060	0.00952	0.006739	0.006249	**0.005086**

TABLE III: CAR TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.02$.

Car				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_001096	0.009414	0.007283	0.009541	**0.007176**
model_0002211	0.009288	0.007016	0.007817	**0.005669**
model_002988	0.010157	0.007051	0.007793	**0.005976**
model_004618	0.009796	0.007378	0.007732	**0.006265**
model_009175	0.010911	0.008789	0.01092	**0.008767**
model_020513	0.009728	0.007317	0.008953	**0.007138**
model_021318	0.009947	0.007129	0.008292	**0.006390**
model_039792	0.010131	0.006880	0.008023	**0.006149**
model_043510	0.010083	0.007716	0.009729	**0.007565**
model_044420	0.010051	0.007271	0.008735	**0.006635**

TABLE IV: CHAIR TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.02$.

Chair				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_002602	0.010282	0.007583	0.006740	**0.006211**
model_005508	0.010152	0.007256	0.007842	**0.006793**
model_008114	0.010432	0.007821	0.008380	**0.007036**
model_014993	0.010784	0.007965	0.006952	**0.006467**
model_017670	0.010684	0.007894	0.008055	**0.006793**
model_022491	0.010424	0.007902	**0.007554**	0.007648
model_039695	0.011569	0.008108	0.009014	**0.006487**
model_042555	0.010087	**0.007619**	0.008246	0.007648
model_044466	0.010623	0.008095	0.006204	**0.004595**
model_049987	0.009745	0.007236	0.008232	**0.007227**

TABLE V: LAMP TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.02$.

Lamp				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_001682	0.009445	0.007927	**0.005579**	0.006005
model_001821	0.010817	0.008352	0.008466	**0.005646**
model_006388	0.009392	0.008274	0.006045	**0.005988**
model_014733	0.009733	0.008468	0.007239	**0.005218**
model_015980	0.008610	0.007243	0.005429	**0.005160**
model_027566	0.008063	0.006922	**0.005277**	0.006450
model_027833	0.009990	0.008928	0.007990	**0.003452**
model_030995	0.009267	0.008065	0.006349	**0.005404**
model_046317	0.011031	0.007982	0.004864	**0.004792**
model_048527	0.010961	0.007739	0.006688	**0.005411**

TABLE VI: PILLOW TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.02$.

Pillow				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_001024	0.011881	0.007448	0.007472	**0.005202**
model_003839	0.011492	0.007652	0.007206	**0.005278**
model_012495	0.010478	0.007599	0.006926	**0.005012**
model_015547	0.011244	0.007752	0.00709	**0.005095**
model_018375	0.011871	0.007406	0.007382	**0.004979**
model_019730	0.011936	0.007311	0.007624	**0.005120**
model_020595	0.010800	0.007841	0.008739	**0.006460**
model_027534	0.010878	0.007924	0.007523	**0.006310**
model_028384	0.011332	0.007549	0.006639	**0.005243**
model_035045	0.01131	0.007891	0.006135	**0.005370**

TABLE VII: RIFLE TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.02$.

Rifle				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_002980	0.006804	0.005897	0.004607	**0.003962**
model_006695	0.010485	0.007897	0.006605	**0.005813**
model_013613	0.008701	0.007431	**0.005565**	0.006480
model_015732	0.009340	0.007797	**0.005465**	0.006985
model_025491	0.008957	0.005897	**0.005365**	0.005742
model_034448	0.007649	0.007897	**0.005445**	0.006036
model_036775	0.009966	0.007431	**0.005815**	0.006460
model_039833	0.008168	0.007797	**0.005353**	0.005697
model_042254	0.008131	0.007596	0.005789	**0.004493**
model_044289	0.009169	0.006415	**0.005621**	0.006872

TABLE VIII: SOFA TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.02$.

Sofa				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_003801	0.010484	0.008274	0.009275	**0.006563**
model_004439	0.009515	0.007312	0.006411	**0.005604**
model_006149	0.009437	0.007510	0.008397	**0.007068**
model_09749	0.010993	0.008737	0.010912	**0.008312**
model_010543	0.010958	0.007960	0.008881	**0.007398**
model_022992	0.011716	0.007662	0.008785	**0.005891**
model_035915	0.011521	0.008918	0.010647	**0.008386**
model_041298	0.008940	0.007542	**0.005913**	0.006830
model_042238	0.010782	0.008317	0.010042	**0.007509**
model_048440	0.011509	0.007814	0.010334	**0.006829**

TABLE IX: SPEAKER TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.02$.

Speaker				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_001031	0.011605	0.008948	0.012321	**0.009635**
model_007663	0.011398	0.007568	0.010699	**0.006660**
model_008001	0.011705	**0.008430**	0.011304	**0.007673**
model_013073	0.012051	0.007312	0.009543	**0.005797**
model_021870	0.011140	0.007515	0.007913	**0.005909**
model_036904	0.012007	0.007351	0.009715	**0.005696**
model_043338	0.011774	0.007391	0.011380	**0.006690**
model_048797	0.012235	0.007995	0.010275	**0.006907**
model_049049	0.011598	0.008055	0.011599	**0.007589**
model_050580	0.011491	0.008178	0.01089	**0.007547**

TABLE X: TABLE TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.02$.

Table				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_000287	0.010379	**0.007332**	0.010276	0.007662
model_000585	0.010493	**0.007786**	0.010412	0.008122
model_001276	0.011480	0.007817	0.009086	**0.007441**
model_006528	0.009003	0.007076	0.008329	**0.006787**
model_011565	0.009279	0.007027	**0.006684**	0.006265
model_017383	0.010297	0.007482	0.009406	**0.006962**
model_021726	0.008840	0.006893	0.006376	**0.005856**
model_028591	0.010119	0.007731	0.009910	**0.007430**
model_047791	0.009644	0.006701	0.006874	**0.006391**
model_048607	0.009726	0.006790	0.006120	**0.005732**

TABLE XI: CATEGORY MEAN FOR SIMULATION FOR WHITE NOISE WITH $\sigma = 0.02$.

Category	GLR	PointCleanNet	GraphPointNet
Airplane	0.007470	0.006253	**0.005491**
Bench	0.007329	0.006481	**0.005422**
Car	0.007383	0.008754	**0.006774**
Chair	0.007748	0.007722	**0.006691**
Lamp	0.007990	0.006393	**0.005353**
Pillow	0.0076373	0.007274	**0.005407**
Rifle	0.007268	**0.005563**	0.005855
Sofa	0.008005	0.008960	**0.007039**
Speaker	0.007874	0.010561	**0.007011**
Table	0.007264	0.008348	**0.006865**

In Table XI the mean over all the models of each category is shown, in order to easily understand which method achieves on average the best results.

It can be seen that GraphPointNet achieves the best results in almost all the categories. The only exception is the Rifle category, where our graph-network provides a denoised version of the point clouds slightly worse than the one denoised by PointCleanNet; however it is able to outperforms the GLR method's results. It has to be noticed that the improvement in the performance of our network with the respect to the other two methods is particularly high at this level of noise, highlighting the advantages brought by the insertion of graph convolution layers.

TABLE XII: COMPARISON BETWEEN GRAPHPOINTNET WITH 66 AND 120 LEARNING FEATURES.

Category	66 Features	120 Features
Airplane	0.005491	0.005434
Bench	0.005422	0.005346
Car	0.006774	0.006783
Chair	0.006691	0.006548
Lamp	0.005353	0.005105
Pillow	0.005407	0.005210
Rifle	0.005855	0.005535
Sofa	0.007039	0.006945
Speaker	0.007011	0.006951
Table	0.006865	0.006904

Enlarging the number of features is a possible improvement of the network to further increase the performance. Therefore, a network with the same architecture of the one presented in the previous simulations but with the number of features incremented from 66 to 120 is trained and tested. A comparison between the original net with 66 features and the one with 120 trained with corrupted data at high level of noise ($\sigma = 0.02$) is analyzed and reported in Table XII.

Expanding the number of features, the network is able to extract more information from the input data, therefore an increasing of the performance is expected. It can be seen from Table XII that on average an improvement is registered. Only in two categories a slightly degradation of the performance can be noticed; even with this, the proposed network still achieves the best performance with respect to GLR and PointCleanNet. The reduction in terms of C2C can be attributed to overfitting phenomena.

A more relevant observation regards the Rifle category. Increasing the number of feature GraphPointNet is able to provide the best performance also in the Rifle category, where with a lower number feature, as shown in Table XI, PointCleanNet is more effective.

5.1.2 Qualitative Results

As additional comparison, qualitative results of the denoising methods are reported. In Figure 19 the original point cloud of an airplane, its noisy version, corrupted by additive white noise with standard deviation equal to 0.02, and the different denoised versions of the airplane are shown.

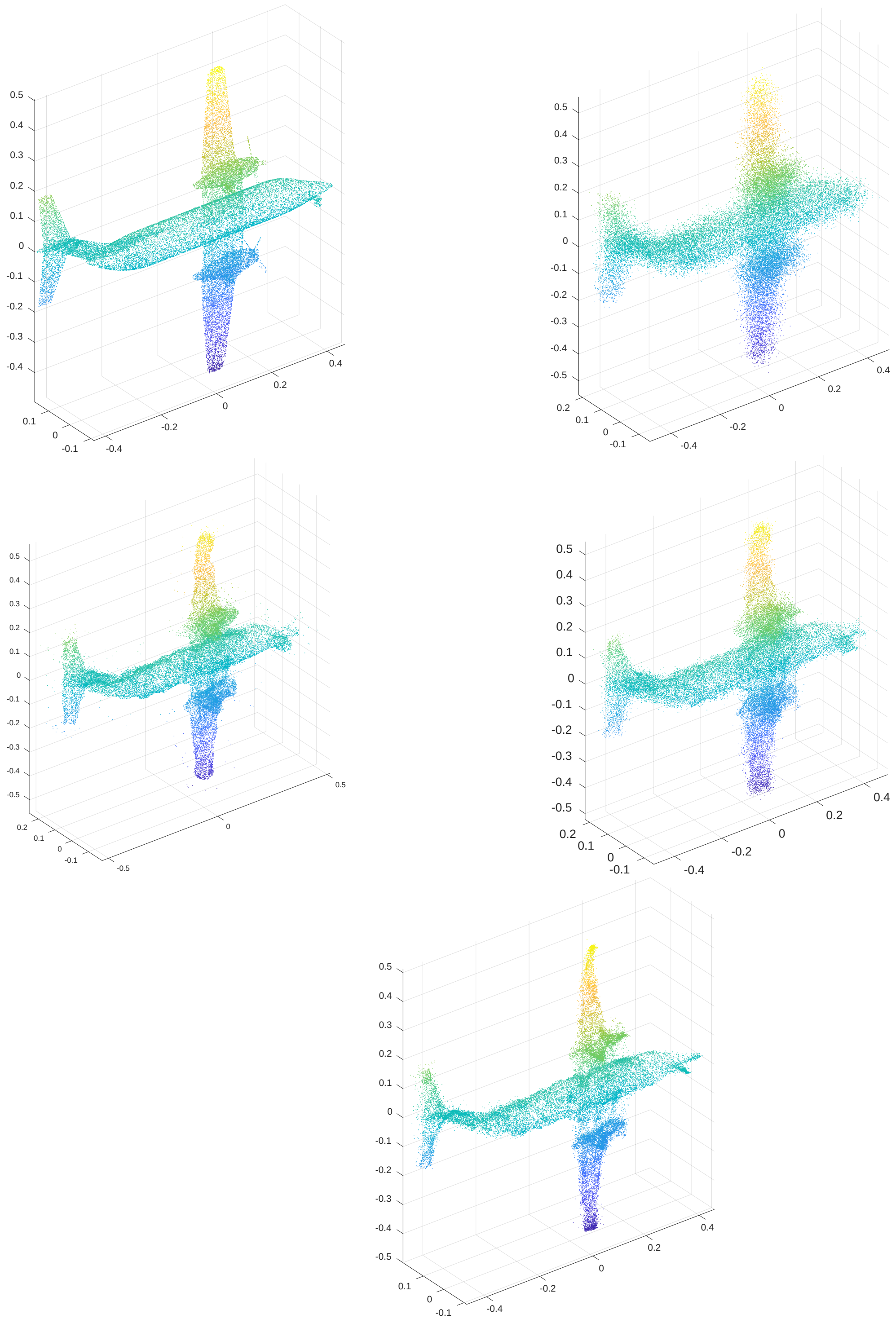


Figure 19: Airplane point clouds. First row: original (left) and corrupted (right) by white noise with σ= 0.02. Second row: PointCleanNet (Left), GLR (Right). Third row: GraphPointNet.

In the denoised point cloud provided by PointCleanNet the presence of several outliers that surround the whole shape of the point cloud can be seen. It is clear that this method is not able to efficiently project all the points upon the point cloud surfaces at high level of noise. Instead, the GLR algorithm is not able to effectively recover details of the global shape of the point cloud: the denoised version remains wider than the original one and the details are not well defined. GraphPointNet is able to sufficiently reconstruct the shape of the original point cloud, with fewer outliers points. In particular our network is able to well reconstruct several details of the original point cloud such as the head of the plane, as visible in Figure 19.

5.2 Simulations with noisy point clouds at low level of noise

5.2.1 Quantitative Results

A further simulation concerning point clouds affected by additive white noise with standard deviation equal to 0.01 is performed. This test is performed over a GraphPointNet model trained in the same conditions of the previous one reported.

TABLE XIII: AIRPLANE TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.01$.

Airplane				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_000492	0.005592	**0.003385**	0.003570	0.003551
model_003733	0.005168	0.003358	0.003583	**0.002564**
model_006263	0.005766	0.003799	0.004015	**0.003766**
model_022125	0.006631	0.005343	0.00637	**0.005135**
model_022283	0.005830	**0.003933**	0.004498	0.003939
model_023833	0.005484	**0.003630**	0.004018	0.003753
model_026886	0.005739	**0.003823**	0.004157	0.003919
model_031422	0.005982	0.004391	0.004737	**0.004159**
model_034021	0.005610	0.003559	0.003759	**0.003351**
model_044620	0.005267	**0.003695**	0.004382	0.003741

TABLE XIV: BENCH TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.01$.

Bench				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_005965	0.006597	0.005010	0.005068	**0.004885**
model_016245	0.005822	0.003370	0.003526	**0.003211**
model_022257	0.005916	0.003678	0.003515	**0.003655**
model_033008	0.005205	**0.004201**	0.005584	0.004603
model_033970	0.005690	**0.004097**	0.004575	0.004400
model_035602	0.005917	**0.003836**	0.003534	0.003843
model_040561	0.005459	**0.004337**	0.005323	0.004590
model_040935	0.005304	**0.003571**	0.004062	0.003966
model_048967	0.005827	**0.003839**	0.004188	0.004111
model_050060	0.005640	**0.004285**	0.005066	0.004667

TABLE XV: CAR TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.01$.

Car				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_001096	0.006849	0.005803	0.006687	**0.005422**
model_002211	0.006033	0.004649	0.005252	**0.004267**
model_002988	0.006588	0.004525	0.005561	**0.004463**
model_004618	0.006326	0.004410	0.004892	**0.004242**
model_009175	0.008037	0.006970	0.007712	**0.006584**
model_020513	0.006876	0.005732	0.006717	**0.005514**
model_021318	0.006650	0.004784	0.005832	**0.004714**
model_039792	0.006751	**0.004644**	0.005633	0.004700
model_043510	0.007111	0.005801	0.006716	**0.005401**
model_044420	0.006797	0.004945	0.005889	**0.004848**

TABLE XVI: CHAIR TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.01$.

Chair				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_002602	0.006392	**0.004401**	0.004486	0.004603
model_005508	0.006533	**0.004911**	0.005461	0.005052
model_008114	0.006850	**0.004863**	0.005721	0.004933
model_014993	0.006687	**0.004199**	0.004419	0.004202
model_017670	0.006868	0.004644	0.005517	**0.004632**
model_022491	0.006910	0.004802	0.004734	**0.004551**
model_039695	0.007308	0.004469	0.005394	**0.004262**
model_042555	0.006843	0.005258	0.005260	**0.005012**
model_044466	0.006031	0.003441	**0.003344**	0.003685
model_049987	0.006793	**0.005652**	0.007685	0.005720

TABLE XVII: LAMP TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.01$.

Lamp				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_001682	0.005727	0.003700	**0.003238**	0.003636
model_001821	0.005960	**0.004039**	0.005225	0.004223
model_006388	0.005358	0.003285	**0.002797**	0.003546
model_014733	0.005327	0.003332	0.003508	**0.002831**
model_015980	0.005123	0.003386	**0.003202**	0.003564
model_027566	0.005232	**0.003845**	0.004318	0.003973
model_027833	0.004921	0.003228	0.003354	**0.002072**
model_030995	0.005378	0.003411	**0.003246**	0.003468
model_046317	0.006705	**0.003320**	0.00338	0.003377
model_048527	0.006874	**0.003907**	0.004095	0.003716

TABLE XVIII: PILLOW TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.01$.

Pillow				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_001024	0.007435	**0.003885**	0.004945	0.004044
model_003839	0.007106	**0.003625**	0.004165	0.003840
model_012495	0.006422	**0.003614**	0.004047	0.003620
model_015547	0.006901	**0.003638**	0.003973	0.003668
model_018375	0.007322	**0.003631**	0.004633	0.003864
model_019730	0.007409	**0.003698**	0.004876	0.004002
model_020595	0.007069	0.004497	0.005624	**0.004401**
model_027534	0.006962	0.004244	0.004825	**0.004091**
model_028384	0.007101	**0.003761**	0.004232	0.003825
model_035045	0.007011	**0.003595**	0.003886	0.003607

TABLE XIX: RIFLE TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.01$.

Rifle				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_002980	0.003869	0.002941	**0.002839**	0.003148
model_006695	0.006620	**0.003906**	0.004082	0.004002
model_013631	0.005499	0.003719	**0.003339**	0.003697
model_015732	0.006059	0.003949	**0.003183**	0.003791
model_025491	0.005434	0.003408	**0.003102**	0.003482
model_034448	0.005101	**0.003958**	0.004034	0.004036
model_036775	0.006315	0.003744	**0.003602**	0.003735
model_039833	0.005024	0.003573	**0.003319**	0.003711
model_042254	0.004462	0.003129	**0.002856**	0.003396
model_044289	0.005970	0.004087	**0.003637**	0.003846

TABLE XX: SOFA TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.01$.

Sofa				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_003801	0.006857	0.005231	0.006073	**0.004152**
model_004439	0.006114	**0.004144**	0.004674	0.004236
model_006149	0.006517	0.005243	0.005717	**0.004876**
model_009749	0.007974	0.006748	0.007798	**0.006228**
model_010543	0.007327	0.004898	0.005506	**0.004676**
model_022992	0.007504	**0.004224**	0.005689	0.004226
model_035915	0.007929	0.005747	0.006959	**0.005245**
model_041298	0.005948	0.004090	**0.003604**	0.003993
model_042238	0.007339	0.005606	0.006768	**0.005279**
model_048440	0.007614	0.005020	0.006635	**0.004884**

TABLE XXI: SPEAKER TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.01$.

Speaker				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_001031	0.008568	0.007517	0.00859	**0.006700**
model_007663	0.007652	0.005363	0.006987	**0.005124**
model_008001	0.008016	0.006200	0.007498	**0.005647**
model_013073	0.007671	**0.004356**	0.006105	0.004461
model_021870	0.007087	0.004316	0.004958	**0.004029**
model_036904	0.007637	**0.004443**	0.006060	0.004517
model_043338	0.007787	0.005448	0.007594	**0.005204**
model_048797	0.007913	**0.004629**	0.006579	0.004757
model_049049	0.008012	0.006124	0.007741	**0.005787**
model_050580	0.007672	0.005489	0.007043	**0.005247**

TABLE XXII: TABLE TESTSET CORRUPTED BY WHITE NOISE WITH $\sigma = 0.01$.

Table				
Model	Noisy	GLR	PointCleanNet	GraphPointNet
model_000287	0.007309	**0.006072**	0.007894	0.006170
model_000585	0.007697	0.006773	0.008212	**0.006635**
model_001276	0.007653	0.005577	0.006659	**0.005398**
model_006528	0.006379	0.005364	0.006648	**0.005260**
model_011565	0.006036	**0.004703**	0.006793	0.005168
model_017383	0.007138	**0.005888**	0.006483	0.005923
model_021726	0.005665	**0.004434**	0.005372	0.004864
model_028591	0.007178	**0.006230**	0.007551	0.006250
model_047791	0.006474	**0.005259**	0.006274	0.005531
model_048607	0.006123	**0.004804**	0.005144	0.005196

TABLE XXIII: CATEGORY MEAN FOR SIMULATION FOR WHITE NOISE WITH $\sigma = 0.01$.

Category	GLR	PointCleanNet	GraphPointNet
Airplane	0.003892	0.004309	**0.003788**
Bench	**0.004022**	0.004444	0.00419
Car	0.005226	0.006089	**0.005016**
Chair	**0.004664**	0.005202	0.004666
Lamp	0.003545	0.003636	**0.003441**
Pillow	**0.003819**	0.004520	0.003897
Rifle	0.003641	**0.003399**	0.003685
Sofa	0.005095	0.005942	**0.00478**
Speaker	0.005389	0.007692	**0.005148**
Table	**0.005510**	0.006703	0.005640

As previously, the Table XXIV shows the mean per category for a better comparison between the methods analyzed.

TABLE XXIV: CATEGORY MEAN FOR SIMULATION FOR WHITE NOISE WITH $\sigma = 0.01$.

Category	GLR	PointCleanNet	GraphPointNet
Airplane	0.003892	0.004309	**0.003788**
Bench	**0.004022**	0.004444	0.00419
Car	0.005226	0.006089	**0.005016**
Chair	**0.004664**	0.005202	0.004666
Lamp	0.003545	0.003636	**0.003441**
Pillow	**0.003819**	0.004520	0.003897
Rifle	0.003641	**0.003399**	0.003685
Sofa	0.005095	0.005942	**0.00478**
Speaker	0.005389	0.007692	**0.005148**
Table	**0.005510**	0.006703	0.005640

Analyzing the results shown in Table XXIV, it can be seen that GLR and PointCleanNet become more competitive, but our network achieves the best results in the majority of the categories.

Notice that at this level of noise the GLR method obtains promising results, with performance similar to the proposed method. This optimization method achieves particularly good results if the point cloud is corrupted with a low level of noise, otherwise, as shown in the previous experiments, it is not able to provide denoised point clouds able to compete with the results of the other methods considered. Instead, GraphPointNet is able to produce denoised point clouds with performances very close to GLR and to outperform PointCleanNet.

For most of the categories taken into account the GLR method and GraphPointNet yield denoised point clouds with close performance in terms of C2C, characterized by a slightly predominance of GraphPointNet.

It is visible from the simulations reported in Table XI that at high level of noise GraphPointNet achieves the best results overcoming all the other methods with a large margin. Otherwise, at lower level of noise it still provides a good denoised version of point clouds, comparable to the results provided by the state-of-the-art, but the gain margins are reduced. The other denoising methods become more effective at low level of noise, providing a competitive denoised version of point clouds, instead our network is able to produce promising results at all level of noise.

CHAPTER 6

CONCLUSIONS AND FURTHER DEVELOPMENTS

6.1 Conclusion

An innovative convolutional neural network designed for point cloud denoising is presented. The novelty lies in the introduction of the graph-convolutional layers, which exploit the Edge Conditioned Convolution, a general graph convolution formulation.

The network is able to outperform the current state-of-the-art and in particular, it achieves promising results when the original point cloud is corrupted by a high level of noise, as presented in Table XI, where the network provides the best denoised point cloud in almost all the categories selected for the testset. Concerning point clouds corrupted by a lower additive white noise, the GLR method becomes more effective and competitive, achieving similar performance to that of our network, as shown in Table XXIV. The proposed method obtains the best performance in the majority of the category. It is clear that the extremely good results are obtained thanks to the graph convolutional structure of the network.

6.2 Future works

As previously discussed, GraphPointNet is able to compete with the state-of-the-art point cloud denoising methods, even with its simple design.

During the training, the network learns to improve the denoised point clouds just having information about the closeness of the estimated points with respect to original clean one, exploiting a generic loss function as the mean squared error and not one specifically designed for the point cloud denoising. In fact, it is not necessary to recover the exact position of each point to obtain a high quality denoised point cloud, but it is important to move the noisy points uniformly distributed on the original surfaces of the point cloud.

Therefore, it could be useful to include this information in the network and insert a block specialized in surface estimation to constrain the denoised points to lie on the point cloud surfaces. In particular, it is possible to estimate the normal at the surface for each of the denoised point, compare it with the original normal and force the network to decrease the discrepancy. The new loss can be obtained adding a regularization term to the original mean squared error that takes into account the normals of the points and minimize the difference between the estimated values and the original ones. This improvement would produce a network able to better estimate the denoised point cloud avoiding outliers and increasing the performance.

CITED LITERATURE

1. Martin Simonovsky and Nikos Komodakis: Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. ArXiv e-prints, 2017.

2. Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P.: The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Process. Mag., 30(3):83–98, 2013.

3. Dong, X., Thanou, D., Frossard, P., and Vandergheynst, P.: Learning laplacian matrix in smooth graph signal representations. IEEE Trans. Signal Processing, 64(23):6160–6173, 2016.

4. David K. Hammond, Pierre Vandergheynst, R. G.: Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis, Elsevier, 30(2):p.129–150, 2011.

5. Qi, C. R., Su, H., Mo, K., and Guibas, L. J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. arXiv preprint arXiv:1612.00593, 2016.

6. Rakotosaona, M.-J., La Barbera, V., Guerrero, P., Mitra, N. J., and Ovsjanikov, M.: Pointcleannet: Learning to denoise and remove outliers from dense point clouds. Computer Graphics Forum, 2019.

7. Chaojing Duan, Siheng Chen, J. K.: 3d point cloud denoising via deep neural network based local surface estimation. Computer Vision and Pattern Recognition, 2019.

8. Guerrero, P., Kleiman, Y., Ovsjanikov, M., and Mitra, N. J.: PCPNet: Learning local shape properties from raw point clouds. Computer Graphics Forum, 37(2):75–85, 2018.

9. Zeng, J., Cheung, G., Ng, M., Pang, J., and Yang, C.: 3d point cloud denoising using graph laplacian regularization of a low dimensional manifold model. arXiv preprint arXiv:1803.07252, 2018.

CITED LITERATURE (continued)

10. Zhang, K., Zuo, W., Chen, Y., Meng, D., and Zhang, L.: Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. IEEE Transactions on Image Processing, 26(7):3142–3155, 2017.

11. He, K., Zhang, X., Ren, S., and Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

12. Jianxiong Xiao, Z. S. A. L. X.: 3D ShapeNets: A Deep Representation for Volumetric Shapes. Proceedings of 28th IEEE Conference on Computer Vision and Pattern Recognition, 2015.

13. Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Fisher Yu, Qixing Huang, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song,, H. J. L. F.: ShapeNet: An Information-Rich 3D Model Repository. Proceedings of 28th IEEE Conference on Computer Vision and Pattern Recognition, 2015.

14. Pistilli, F.: GraphPointNet: Graph Convolutional Neural Network for Point Cloud Denoising. Master's thesis, Politecnico di Torino, 2019.

VITA

NAME	Francesca Pistilli
EDUCATION	
	Master of Science in “Electrical and Computer Engineering, University of Illinois at Chicago, March 2020, USA
	Specialization Degree in “ Electronic Engineering ”, Oct 2019, Polytechnic of Turin, Italy
	Bachelor’s Degree in Electronic Engineering, Sept 2017, Polytechnic of Turin, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2017 - IELTS examination (7.0/9)
	A.Y. 2018/19 One Year of study abroad in Chicago, Illinois
	A.Y. 2017/18. Lessons and exams attended exclusively in English
WORK EXPERIENCE AND PROJECTS	
2018	Top-down design of a custom DLX processor
	Design of digital micro and macro-architectures: design, VHDL description, simulation, synthesis, place & route of DLX microprocessor in all its part starting from scratch, pipelined version, data and control hazard management.
2018	Design of AES Hardware Trojan
	Group Project finalized to the development of two different Hardware Trojan to attack an AES Encryption system. The first Trojan consist in the insertion of a capacitor used stealthy force a certain value to the encryption key; the second, exploits a non-reliable realization of a Control Unit to insert a malicious state, reached through the insertion of a Clock signal whose frequency violates the required time constraints.
2017-2019	Other Experiences:

VITA (continued)

Use of prototyping boards, micro controllers, FPGAs, inside a bigger system or stand-alone (experience: realization of some measuring and control systems, simulations of designed components on FPGA)

Development of all the parts of the software needed to realize measuring systems, data processing systems or mechanical control

Scripts for simulation and synthesis of integrated circuits

Multi-process and multi-thread C programs (in particular for encryption)