POLITECNICO DI TORINO

Corso di laurea in Ingegneria Elettronica

Tesi di laurea Magistrale

# GraphPointNet: Graph Convolutional Neural Network for Point Cloud Denoising

Advisor:
Prof. Maurizio Martina
Prof. Enrico Magli

Candidate:

Francesca Pistilli

Ottobre 2019

# Summary

The project proposed is finalized to develop a novel network for point cloud denoising based on graphs.

A point cloud is an object representation composed by a collection of 3-D space coordinates. This data is usually acquired by radar, laser, electro-optical systems or by reconstruction of 2-D images: all these methods lead to point clouds typically affected by noise. The aim of the project is to design a network able to efficiently re-produce cleaned 3-D point cloud from a noisy observation.

Denoising task is a typical problem addressed in Image Processing and the current state-of-the-art are Convolutional Neural Networks (CNN), that leads to promising results for noise removal in images; the idea developed in this project is to exploit a deep neural network structure composed by convolutional layers, introducing appropriate adjustment for the point cloud denoising task.

The novelty of the project is the introduction of a graph-convolutional layer, that exploits the Edge-Conditioned-Convolution [1](ECC) to implements a graph-convolution operation over point cloud.
A graph is computed for each point cloud, where each single point is a node and the weighted connections between them are the edges. 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 vector at layer $l$ of the node itself and the nodes in the neighborhood.

Traditional methods to denoise a point cloud are geometrical algorithms, that can be based on graph representation. Some popular projects involve the computation of surfaces of the point cloud from the noisy observation and then the projection of the noisy points, instead others represent the point cloud on graph and exploit graph-regularization method. All these approaches leads to a classical optimization problem.

Recently, due to the increasing interest in the point cloud denoising, new approaches are explored, in particular several neural network project able to outperform the traditional methods have been published.
None of the 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.
The network proposed in this thesis, called GraphPointNet, would be the first neural network based on convolution able to process point cloud.

In this thesis after an introduction section where the basic concepts of neural network and graph theory are presented, the current state-of-arts are summarized. 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 evaluations of the network proposed are reported. Quantitative and qualitative test are performed in order to evaluate the results obtained. In particular, the point-to-point distance is taken into account to evaluate the goodness of the results obtained and to make comparisons with other methods. It is shown that the method proposed is able to outperform or at least match the current state-of-arts.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In the introduction chapter the background concepts of the project are presented, starting from a brief description of the input data, the point cloud, and following with a overview of neural network and graph theory.

## 1.1 Point Cloud

3D point cloud are collections of data points that represent cities, environments, artificial systems or objects of all dimensions. The data is expressed by geometrical coordinates $(x, y, z)$ of sampled points from the surface of the analyzed shape.

Point cloud are becoming increasingly popular because of the ability to provide a detailed representation of the real world and the wide applications in many different areas, such as architecture, medical imaging, virtual reality, and aeronautics applications.

Recently, there is a growing interest regarding the acquisition and processing of point clouds: new techniques to increase the quality of the data and the possible applications are investigated. Different approaches to the point cloud acquisition are presented in literature as electro-optical systems, laser scanning or radar system, reconstruction starting from 2-D images.

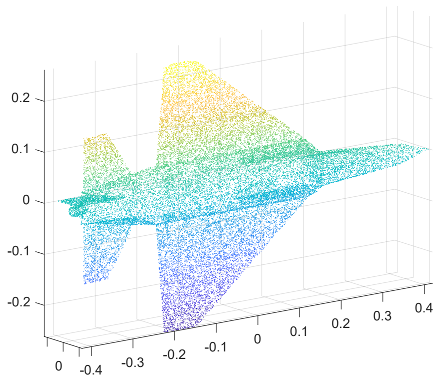


Figure 1.1: Example of Point Cloud

## 1.2 Neural Network

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

The basic element of a neural network is a *perceptron*. The general idea is to emulate the biological neurons: several signals arrive at the nerve cells, where are processed and the cells eventually produces a response.
A perceptron is a mathematical algorithm, that takes as inputs vectors of numbers, weighs them element by element, sums them together and apply at the results a nonlinear function to obtain output. The output is a binary classifier, that would make a decision, True or False, according to the weighted input. An example of perceptron is reported in Fig.1.2.

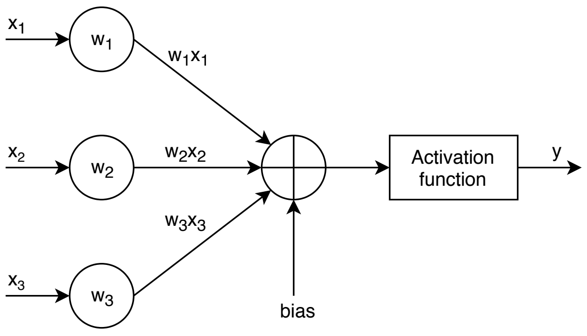


Figure 1.2: Perceptron

In Fig. 1.2. can be seen that the weighted input are summed together with a bias, that is a scalar quantity inserted to move the decision threshold far from the origin.

The perceptron was originally designed for binary classification task, as image recognition. At first, the nonlinear function, also called *activation function*, was a simple function with only two possible output, 0 or 1, i.e. True or False.

Afterwards, the neural network are exploited in more complex tasks, to find functions able to map inputs of the dataset to the correct outputs. The operation involved became gradually more complicated and the activation function and the structures consequently.

If several perceptrons are connected together, a neural network is created, see Fig.1.3. as example. The network can be more or less *deep* according to the number of hidden layer inserted and the layer can be *fully connected*, if each output of intermediate layers is a input of the following layer, or not. How many layers and how connect them are a designer decisions, made taking into account the specific task of the net.

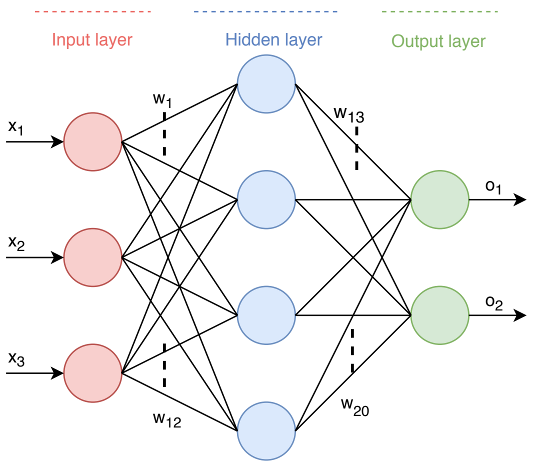


Figure 1.3: Neural Network Example with one hidden layer

In the development of a neural network after the general structure of the net is designed, the weights and the bias involved are initialized and the network has learn the correct values to obtain acceptable results. The initialization process is a complicated topic that would be outside the brief introduction addressed in this section and it can be simplified as a random initialization.
The *training phase* is the learning process where the parameters are changed in order to obtain the desired percentage of right predictions: due to the complexity of the classification a certain failure percentage has always to be taken into account.

To evaluate the network performance a *loss function* is exploited, able to describe the discrepancy between the output of the net and the desired one, a popular function is the mean squared error (MSE) loss, that compute the squared difference between the predicted data and the true one:

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_{true} - y_{predicted})^2 \tag{1.1}$$

where $N$ is the total number of points.

It is easy to understand that the goal of the training phase is to minimize the loss function: to fulfill this task an optimization method applied to the loss function is applied and causes the variation of the network parameters. A popular optimization algorithm is the gradient descendent, that is able to determine how change the parameters computing the partial derivatives of the loss function with respect to each trainable variables.
Considering the variable $w_1$ of the net reported in figure 1.3 the variable update would be:

$$w_{1,updated} = w_1 - \eta\frac{\partial L}{\partial w_1} \tag{1.2}$$

where $\eta$ is the *learning rate*, a parameter that controls how much the variables can change at each updates: a minor the learning rate causes a fine-tuning of the parameters.

During the training phase, a training dataset is exploited. The dataset is composed by a collection of data with the correspondingly true information that has to be predicted. For instance if the net addresses a image classification task, the dataset is a collection of image and the corresponding category associated at each image, called *label* is the information that has to be predicted. Instead regarding a denoising problem, the dataset would be the noisy image that has to be reconstructed and the original clean version is the data estimated by the network. An algorithm where is available the ground truth, the true information that has to be predicted, is called *supervised learning.*

The neural networks are generally characterized by three phases: the training, the validation and the testing. The validation phase consists in momentarily stopping the training and testing the net as far trained with data belonging to the same original dataset of the training dataset, but that are not included in the latter one, therefore the net has never seen during the learning phase the validation data. This operation is performed in order to check if the network is over-adapting the parameters over the training dataset, i.e. the noticed decreasing of the loss is due to a over fitting over the known data and over new inputs the same performance are not met. Only during the training phase the parameters are updated, the validation is a off-running check of the performance. Finally the testing is performed when the network is trained and usually completely different data are involved.

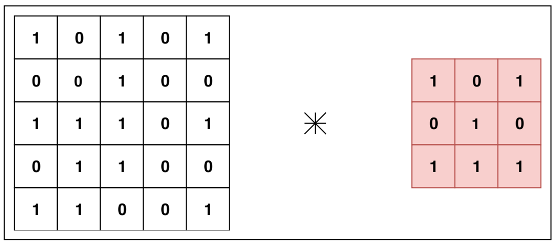
## 1.3 Signal Processing application of Neural Network: Convolutional Neural Network

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 operation involved is provided.

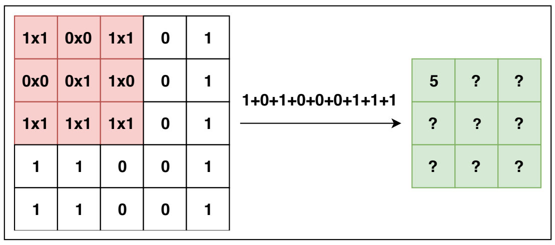
A CNN is a *deep neural network* that exploits the convolution operation instead of simple matrix multiplication between input and weights.
It takes images as input, each one represented by a tensor with dimensions (*height* x *width* x *channels*), where *channels* is the depth of the data: it is equal to 1 for gray-scale or to 3 for RGB images. The output of the network depends on the intended use.
The internal structure is characterized by several hidden layers, consisting in convolutional layers, followed by activation function 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 ($height_{filter}$ x $width_{filter}$ x $channels$), where the parameter *channels* has to be equal to the image's depth. All the elements of the filters consists in the weights of the network that has to be learned.
The network is able to detect and isolate information and features of the data that can be used to fulfill the addressed task. The network convoles the input data with specific filters in order to capture the important characteristics.
A convolutional layer is realized studying and emulating the response of a neuron in the visual cortex to a stimulus. Each neuron is able to processes data only for its receptive field and if several fields are considered it is possible to cover the whole visual area.

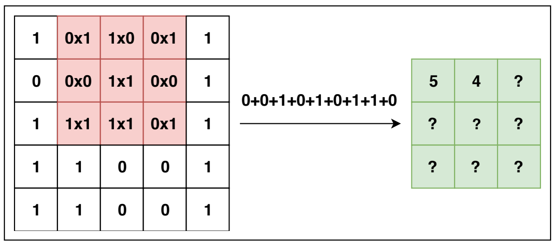
To perform a convolution operation a selected filter slides over the input matrix, isolating windows of data, and merges the data selected with the filter values. A fixed number of shifts is performed to move the window over the matrix and the number of shift is set during the design and it is called *slides*. In Fig.1.4. is reported an example to clarify the operations of a generic CNN.
After the convolutions operations, the modified data are passed as input of the following layer.
During the training phase, the network changes and learns the filter's elements in order to extract meaningful information for the established task.
The power of the architecture is due to the reduction in the number of parameters involved and reuse of weights with respect to other fully connected layers.



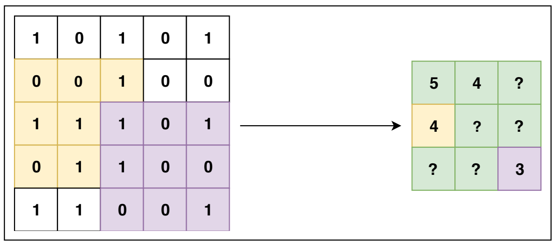
(a) Input matrix 5x5 and filter 3x3



(b) Computation of the element (1,1)



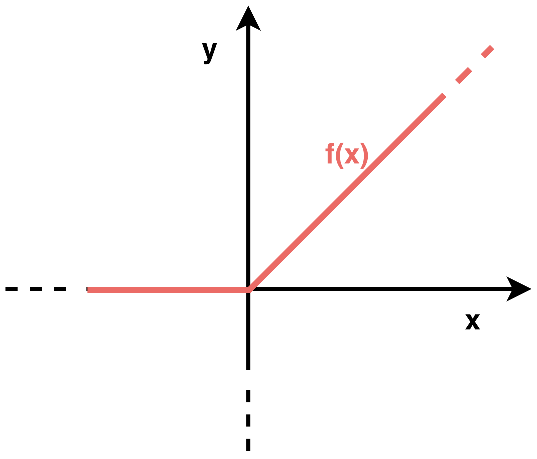
(c) Computation of the element (1,2)



(d) Element (2,1) and (3,3)

Figure 1.4: Convolution Operation Example: In figure (b), (c) and (d) the computations reported regards a convolution operation between the input matrix and the filter in figure (a) with slide equal to 1

An activation function is inserted after each convolution layer, for the same reasons described in the previous section. 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: this type of layer is in charge of reducing the number of parameters, taking into account only the most significant one. For instance, a popular choice is to perform a *Max Pool*:only the maximum parameter over a selected window is considered.
Following is reported an example of a CNN for a classification problem in Fig. 1.5.

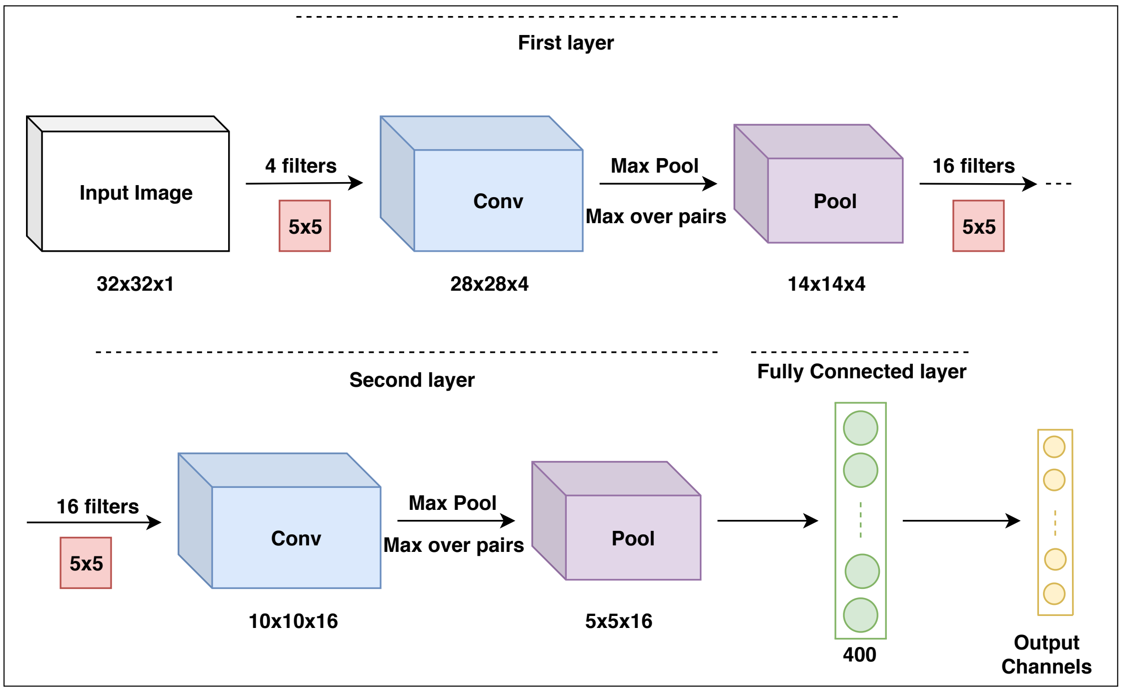


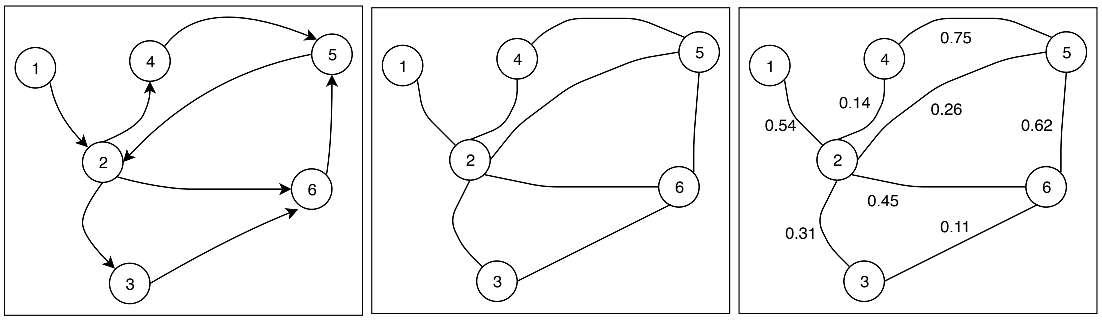
Figure 1.5: CNN Example

A CNN is the state-of-arts in terms of neural network for image processing, due to the high performance achieved, but it is mainly exploited for 2-dimensional inputs. The novel idea presented in GraphPointNet is to introduce a convolutional network in a structure compatible with more than 2-dimension data.

## 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 exploited in the project.

A generic graph $\mathcal{G}$ is constituted by a collection of points, called *vertices* or *nodes*, and the connection between them the *edges*, respectively denoted with symbols $\mathcal{V}$ and $\mathcal{E} \subset \mathcal{V}$ x $\mathcal{V}$.
A graph can be undirected or directed: in the first case, choosing two random vertices 1 and 2, if the vertex 1 is connected to the vertex 2 then the vertex 2 is connected to vertex 1; otherwise the direction of the edge is univocal, specified by an arrow, as shown in Fig.1.6.



(a) Directed Graph (b) Undirected Graph (c) Weighted Graph

Figure 1.6: Graph Examples

Considering a general 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 $A \in N$x$N$, 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.
Following, the adjacency matrix of the undirected graph is shown in Fig.1.6. as an example. It is straightforward to understand that for a undirected graph the adjacency matrix would be symmetric.

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

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

Considering a weighted graph, a *weighted matrix* $\mathbf{W} \in N\text{x}N$, similar to the adjacency matrix, can be defined:

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

Where $w_{ij}$ represent the specific weight related to the edge $e_{ij}$.
The degree matrix related to the weighted graph in Fig. 1.6c. 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 and weighted 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 $i$:

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

The degree matrix build upon the third graph of Fig. 1.6c. is:

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

If a directed graph is considered the direction of the edge is taken into account accordingly associating different sign to edge in different directions.

Finally, the *Laplacian matrix*, that plays a key role on the 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

Exploiting the representation in frequency domain of a signal in the time domain it is possible to obtain information about the signal itself: it is then interesting to introduce the frequency analysis for graph signals.

In order to obtain information about the frequency of a graph signal it is necessary to exploit the Graph Laplacian matrix, previously introduced. The **L** matrix is symmetric and positive and represents an approximation of the Laplacian operator, therefore it is possible to formulate a Fourier-like Transform.

Given a graph signal $f : \mathcal{V} \to \mathcal{R}^N$, where $\mathcal{V}$ is the set of vertices of the graph, it is possible to define the following formulations:

$$\mathbf{Lf} = \sum_{i,j=0}^{N-1} W_{i,j}(f(i) - f(j)) \tag{1.5}$$

$$\mathbf{f}^T\mathbf{Lf} = \frac{1}{2}\sum_{i,j=0}^{N-1} W_{i,j}(f(i) - f(j))^2 \tag{1.6}$$

The equation 1.5 correspond to a difference operation and the 1.6 is the quadratic form the graph Laplacian matrix that gives information about the smoothness of the signal.

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 $\chi_i$ is a eigenvector of **L** and $\lambda_i$ the corresponding eigenvalue.
The matrix **Q**, where each column is a eigenvector of **L**, is introduced and the matrix **L** can be factorized as:

$$\mathbf{L} = \mathbf{QLQ}^T \tag{1.8}$$

The eigen-decomposition 1.8 can be easily demonstrated by the eigenvector-eigenvalue theory expanding the eigenvalue equation 1.7.

For any signals $f$ in the time domain, the classic Fourier transform and its inverse are defined:

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

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

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

Similarly, for any graph signal $f$ can be defined the graph Fourier Transform:

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

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

The equations of the graph Fourier transform $\mathcal{GFT}$ 1.11 and its inverse $\mathcal{IGFT}$ functions are presented in [5].

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:

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(x-t) g(t) dt \tag{1.13}$$

It is not possible to directly apply the classical formulation to graph signals because in the graph domain the signal translation $f(x-t)$ is not defined.
Therefore, a new expression for convolution over graph is delineate exploiting the graph spectral domain. The equation 1.13 can be re-written as:

$$(f * g)(x) = \mathcal{IGFT}\Big\{\mathcal{GFT}\{f * g\}\Big\} \tag{1.14}$$

The same properties of the Fourier Transform are still valid in this case; in particular, the Graph Fourier Transform of two convolved signal is equal to the multiplication of the Graph Fourier Transforms of the two signals.

$$(f * g)(x) = \mathcal{IGFT}\Big\{\mathcal{GFT}\{f\}\mathcal{GFT}\{g\}\Big\} \tag{1.15}$$

The graph-convolution operation can be defined in the spectral domain applying the equations 1.12 and 1.11:

$$(f * g)(n) = \sum_{l=0}^{N-1} \hat{f}(l) \hat{g}(l) \chi_l(n) \tag{1.16}$$

### 1.4.2 Dynamic Edge-Conditioned Convolution [1]

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

Usually, a convolution operation is applied to data that lies on a regular grids and it is the key operation on which the Convolution Neural Networks (CNNs) relies, as reported in previous section.

Due to the wide applications of the classical convolution operation, it becomes interesting formulate a definition of a convolution operation suitable for any signal, even if represented by irregular structures and lies on non-Euclidean domains, but easily defined on a graph.

In the previous section a graph convolution operation formulated in the spectrum domain, based on the GFT, is presented. The operation is characterized by a high computation cost and it is not suitable with data with a variable structure, such as point cloud, due to the fact that the filters involved are computed in the spectrum of graph laplacian.

The operation presented in [1] is the generalization of the classical convolution to every type of signals formulated in spatial domain.
In [1] the convolution operation is interpreted as a weighted aggregations over a neighborhood of a graph, exploiting a spatial approach instead of a spectral method, leading to consider various types of graphs, with no restriction on the structure and the size.
It is considered a generic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, constituted by $n$ vertices and $m$ edges. In order to implement the ECC a feed-forward neural network, with $l_{max}$ number of layers, is exploited and the considered graph has to be *vertex and edge labeled*, at each node and edge a label is associated according to a specific function: $X^l : \mathcal{V} \to \mathbb{R}^{d_l}$ associate at each vertex a feature and $L : \mathcal{E} \to \mathbb{R}^s$ associated at each edge an attribute.

A key concept in the algorithm is the definition of neighborhoods. A neighborhood $N(i)$ related to the vertex $i$ consists in a collection of adjacent nodes and the node itself: $N(i) = \{j; (j,i) \in \mathcal{E}\} \bigcup \{i\}$.

As previously mentioned, the operation here presented computes the signal $X^l(i)$, related to the $i$ vertex, as a local weighted summation of signals $X^{l-1}(j)$ belonging to its neighborhood. As a matter of fact, the ECC is performed as a local operation, the actual computation is applied to a section of the graph by time and therefore it is possible to manage different structures of graph, any order of the vertices and sizes without any constraints.

A topic worthy of discussion is the definition of the weight matrix $\Theta^l_{ji}$ involved in the local weighted aggregation. It is defined a filer-generating network $F^l : \mathbb{R}^s \rightarrow \mathbb{R}^{d_l x d_{l-1}}$, that takes as input $L(j,i)$ and gives as output the matrix $\Theta^l_{ji} = F^l_{w^l}(L(j,i)) \in \mathbb{R}^{d_l x d_{l-1}}$. It is has to be noted that the term $w^l$ of the function $F^l$ represents the weights of the network that generates the matrix $\Theta$.

In the case under study, the edge labeling function considered consists in the features associated the two nodes composing the edge at each layer $l$:

$$L(i,j) = H^l_j - H^l_i, \; j \in N(i) \tag{1.17}$$

In the previous equation the vector $H^l_i$ represent the feature vector of node $i$ of the $l-th$ layer.
In conclusion is here reported the equation that described the ECC:

$$\begin{aligned} H^{l+1}_i &= \frac{1}{|N(i)|}\sigma\Bigg(\sum_{j \in N(i)} F^l_{w^l}(L(j,i))H^l_j + b^l\Bigg) \\ &= \frac{1}{|N(i)|}\sigma\Bigg(\sum_{j \in N(i)} F^l_{w^l}(H^l_j - H^l_i)H^l_j + b^l\Bigg) \\ &= \frac{1}{|N(i)|}\sigma\Bigg(\sum_{j \in N(i)} \Theta^l_{ji}H^l_j + b^l\Bigg) \end{aligned} \tag{1.18}$$

The symbol $\sigma$ represent a non linear function typically used in Neural Network.

# Chapter 2

# State of the Art on Point Cloud Processing

In literature, different typologies of processing of point cloud are available: projects addressing various tasks such as classification, segmentation and denoising have been developed by several groups [2, 3, 4, 6, 7], confirming the increasing interest towards point cloud and their application.
In this section, particular attention is given to point cloud denoising: the importance of this task, together with the different possible approaches to address it are discussed and the most relevant projects are presented, highlighting the differences and common points among them.

## 2.1 Point Cloud Denoising

Point cloud is a 3-D signal representation, very popular due to the ability to discrete represent surfaces and to its wide applications.

The available acquisition methods, sensors as Microsoft Kinect or algorithms able to create the point cloud from several images, insert non-negligible noise, corrupting the information collected, making necessary a denoising pre-processing before the data application. For all this reasons, the denoising of point cloud has become a problem addressed by several researchers with different methods.

Traditionally approaches can be dived into two main categories: local and non-local operators based. Local methods generally regards the surface fitting to noisy input; one of the most common used is the Moving Least Squares method, that projects each noisy input points to a surface approximated from the noisy observation. Other popular approaches are based on sparse representations of geometric characteristic, such as the surface normals, estimated solving a minimization problem. Concerning the methods belonging to the second category, the non-local approaches are based on self similarities to re-construct the noiseless point cloud. A variant of the last categories are the graph-based approaches, where graph properties are exploited, particularly efficient are the methods that particularity involve signal smoothness and regularization, as Graph Laplacian Regularizer, Graph variant of Total Variation etc.

Recently different techniques to solve the problem have been investigated and the most promising is exploiting neural networks. Few works have been presented, the most relevant are PointCleanNet [3] and a network based upon local surface estimation presented in [4], called Neural Projection Denoising (NPD). Both this networks are based on the structure described in [2], called PointNet: this project is actually the first neural network designed to deal with point cloud with segmentation and classification as addressed task.

None of the neural network proposed for the denoising of point cloud implement a convolution network, it is coarsely emulated using *multiple layer perceptron*, but it is never take into account to create a convolutional network, that is one of the strength and the novelty of GraphPointNet.

## 2.2 Notable Methods for Point Cloud Processing

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

### 2.2.1 PointNet [2]

PointNet is the first neural network able to efficiently deal with point cloud for classification and segmentation task. The architecture presented, shown in Fig. 2.1. just the classification network, is quite simple, characterized by several *multiple layer perceptron*, $MLP$, and *maxpooling* layers.

It can be seen in Fig. 2.1. that two *transformer* blocks are inserted, $STN_1$ and $STN_2$, both of them have an internal architecture similar to the global structure of PointNet: several fully connected layer with a final maxpooling layer. The $STN_1$ block is the input transformer that creates a 3x3 matrix from the nx3 input matrix in order to constrain the network rotation invariant and the second block, $STN_2$, is the feature transformer similar with the previous one but the output is a 64x64 matrix. In particular one transformer consist in shared $MLP$, with output dimensions (64,128,1024), a *maxpooling* and two fully connected layers to regress the dimensions.
It can be noticed that the number of input points and the size of the max layer output have a great impact on the performances of the net, hence it is reasonable

utilize point cloud with a limited number of points.

PointNet is a state-of-the-art neural network designed for point cloud, characterized by a simple architecture, it is able to achieve promising results for the addressed tasks. After the publication of this project, several works that exploit a similar architecture but addressing other processing task has been realized; the most worthy of mention are the *PCPNet* [6] for the estimation of local shape properties, such as the normals of the points, *PointCleanNet* [3], for the denoising of the point cloud, that is inspired by *PCPNet* and *Neural Projection Denoising* [4],*NPD*, based on local surface estimation.
In the following *PointCleanNet* and the *NPD* network are well discussed; otherwise it is not presented an in-depth review of the project PCPNet because the architecture would not introduce any novel aspects and the task is beyond the topic of the thesis.

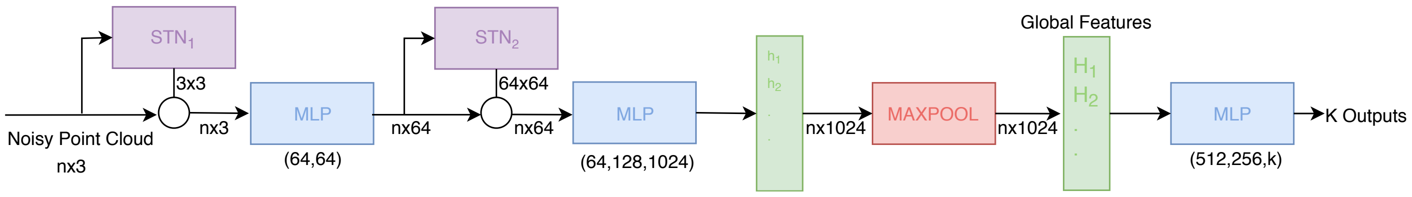


Figure 2.1: PointNet: architecture for Point Cloud classification

### 2.2.2 PointCleanNet [3]

PointCleanNet is a deep neural network that, given a noisy point cloud, is trained to learn surface patches from the features extracted from the corrupted input and exploits them to identify the outliers, additional points that are outside of the original point cloud, and denoise the remaining points affected by white noise, projecting them onto the surface.

The data-set of the training set is a collection of patches, created by selecting a point and considering its closer points in a radius $r$, and the algorithm only perform the denoising of the center point of the patch, that depends only on a local neighborhood, leading to treat the denoising task as a local problem.
The network implemented is divided into two stages: the first is finalized to the outlier removal, where an outlier probability, $o_i$, associated to each points is predicted and if it is larger than 0.5 the points 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 is estimated in applied to the input order to obtain the final denoised point cloud.

The structure exploited in both of the steps can be seen in Fig. 2.2. and Fig. 2.3. It is largely drawn from PCPNet [6], that in turn is inspired from PointNet, but characterized by different loss functions. The goals of the network are move the noisy points close as possible to the original surface and maintain a regular distribution, avoiding clustering, and the loss functions are set accordingly. Several alternatives are tested and the one selected, that achieves better results, exploits the distance between each denoised points and its closer point in the ground truth point cloud to move the point close to the surface, $L_s$ term in equation 2.1, with an additional regularization term $L_r$, to avoid clustering.

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

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

In Fig. 2.2. and Fig. 2.3. is shown the architecture of the network proposed in the paper [3], where *Spatial Transformer* layers, *QSTN* and *STN* blocks, and *fully connected* layers, *FNN*, are exploited. The first layer, *QSTN* is used to constrain the network rotation independent, clearly at the end of network, the estimation has to return to the original rotation to be consistent with the input data. It has the same function of the first transformer block in PointNet, the only difference is the output in PointCleanNet is a quaternion instead of a 3x3 matrix. The subsequent layers are finalized to the feature extraction that is realized with several *fully connected layers* and a full linear transformation, *STN* block. The computed features for 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 by fully connected layers, estimates the desired output, the displacements $d_i$ or the outlier probability $o_i$.

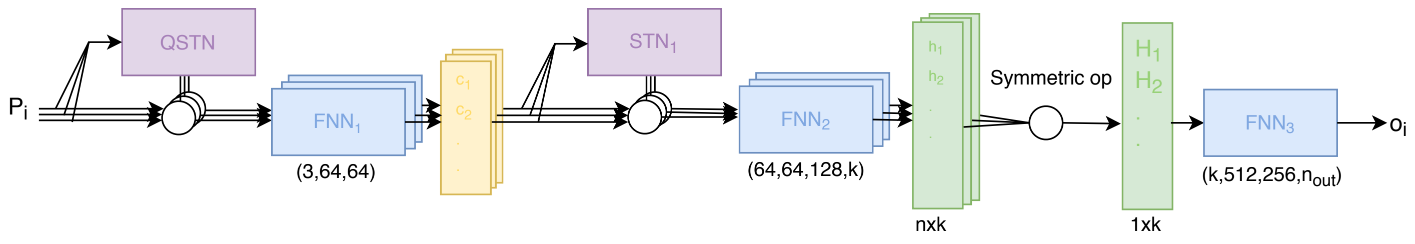


Figure 2.2: PointCleanNet: architecture of the outlier removal step

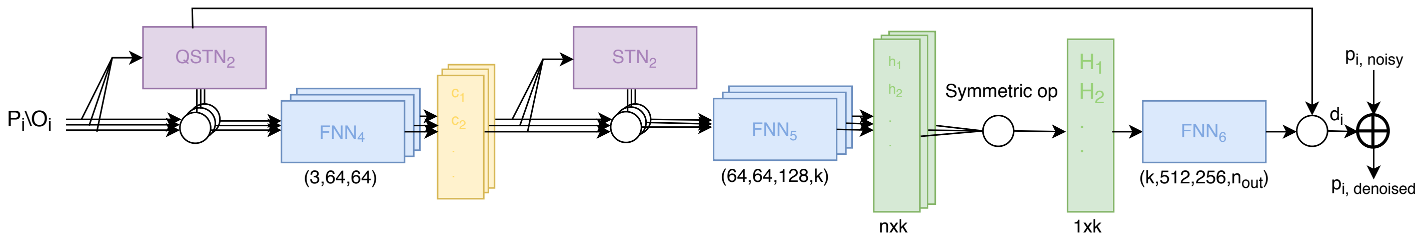


Figure 2.3: PointCleanNet: architecture of the denoising step

PointCleanNet is able to outperforms currently state-of-the-arts traditional methods, exploiting the simple architecture of PointNet re-arranged for the denoising task. Furthermore, it is a blind network: it is not necessary to train the network for a specific type of noise, i.e. with specify the standard deviation of the white noise, but the same network can denoise point cloud affected by a range of white noise. However, the network is able to denoise just the center point of the patch, performing a point training rather than a patch training.

### 2.2.3 3D Point Cloud Denoising via Deep Neural Network Based Local Surface Estimation [4]

The Neural Projection Denoising project, *NPD*, is a novel project that exploits the estimation of geometrical properties to denoise the corrupted point cloud, the input of the net.

The algorithm predicts the reference planes for the noisy input and projects the points to the corresponding surfaces. Therefore, the network has to learn from local and global features the reference planes $\hat{T}_i$, characterized by the normal vector $\hat{\mathbf{a}}_i$ and interception $\hat{c}_i$ for each noisy points $\tilde{\mathbf{p}}_i$ and then obtain the denoised point.

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

During a pre-processing phase, the true reference planes are calculated with graph-based methods from the original point cloud and the results obtained are used to supervise the estimation of the surfaces jointly with the noise-less point cloud that supervise the final denoised version of the data.
Two loss function are exploited, to evaluate the goodness of the final denoised point cloud the *mean squared error*, *MSE*, is calculated:

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

Instead, for the reference plane loss, the cosine similarity is exploited

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

The total loss is the combination of equations 2.4 and 2.5.

The architecture, as already mentioned similar to PointNet, is shown in Fig. 2.4. It can be seen that the *transformer block*, typical of the PointNet architecture, are not reported: it is only necessary a *MLP* to extrapolate the local features and a *maxpooling* operation for the global information. The two features such predicted are concatenated and they are the input of a series of *MLP* that estimates the reference planes. Finally, the noisy points are project on the estimated reference planes, obtaining the denoised results.

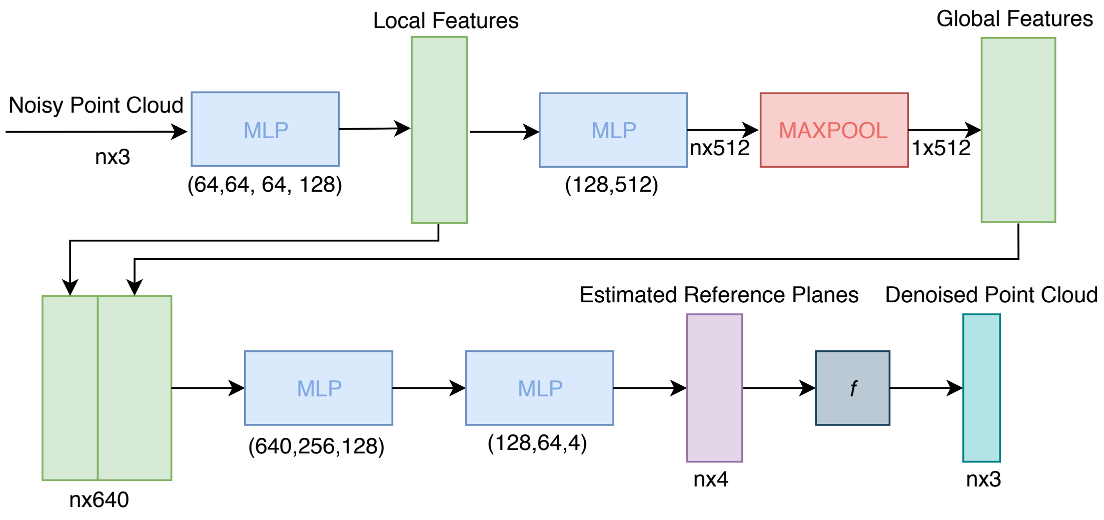


Figure 2.4: Neural Projection Denoising: Architecture of the whole network

As in the previous denoising network, the strength of the project in the simplicity of the architecture: it directly estimates surfaces and project the points, creating a resulting point cloud better than other state-of-arts methods.

# Chapter 3

# Proposed approach for Point Cloud Denoising

In this section different aspects of the design of the network, before the actual development of the architecture, are described, as the general structure, the creation of the data-sets and the handling of data.

## 3.1 General Structure

The first step of the project's design is to delineate a general structure of the network.

As previously discussed, the method proposes a convolutional network, characterized by graph-convolution layers built exploiting the *Edge-Conditioned Convolution* [1] to address the denoising task.

Recently, it has been demonstrated that a Convolution Neural Network for image processing task achieves better results if trained to predict the residual between the noisy input and the ground truth instead of directly estimate the denoised result [8]. The same principle is exploited in this project: the network learns to progressively remove the clean point cloud from the noisy observation. A in-depth discussion of the structure is reported in section 4.2.3.

The body of architecture is composed by several blocks, called *Residual Block*, drawn by the building block presented in [9], able to detect and remove the correlations of the input of the block itself: in this way, starting from the noisy observation, the true point cloud is gradually removed, ideally obtaining at the end of the net only white noise.
The estimation of the noise is subtracted to the noisy input in order to finally obtain the denoised point cloud.

All the graph-convolution operations are performed in the feature space; therefore a *Pre-Processing Block* is necessary to move from the spatial domain, in which the noisy point cloud is represented, to the feature space, extrapolating meaningful information. In Fig. 3.1. it is possible to observe the general architecture described: only two residual blocks are reported as an example, the number of elements inserted in the final net will be discussed in section 4.2.3, where a wider explanation of each block is reported.

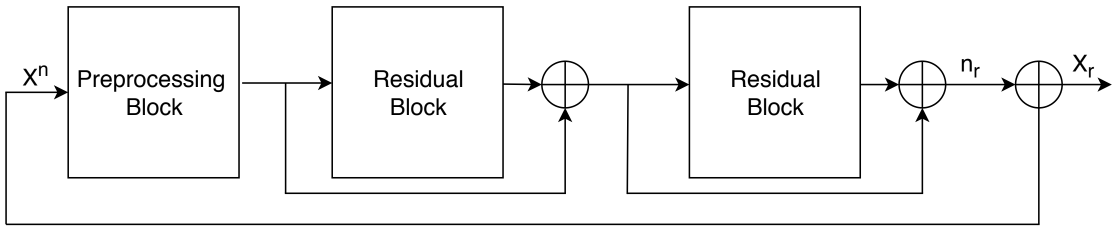


Figure 3.1: Block Diagram of the General Structure proposed

## 3.2 Dataset creation

An importan aspect worthy of discussion is the creation of the data set: for the training and validation set the **ModelNet40** [10] database is exploited; instead for the testing phase point clouds from the **Shapenet** [11] repository are used as inputs.
Both the previous cited archives store 3-D representations of objects belonging to different common categories: airplanes, beds, chairs, baths etc.

ModelNet40 is provided by the University of Princeton and it is constituted by a collection of 3D CAD models for 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 meshes: in each file the *vertices* and the *faces* are reported, the first by the spatial coordinates and the second are expressed as the indices of the vertices that compose them, since they are geometrical portions of the surface. On the other hand, Shapenet, released by the University of Standford, 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 in the same way as the other cited dataset.

All the data that constitutes the training, validation and testing set has to be pre-processed in order to apply the denoising method to a generic point cloud in the same initial conditions. The first step is to perform a mesh-sampling on the faces reported in the original files, in order to obtain a collection of points, identified by their spatial coordinates: for the training/validation data 20.000 points per point cloud are sampled, instead for the testing 30.720. In particular,

to maintain a uniform distribution and avoiding zones of different amount of data, the point are sampled following a normal distribution.
After the sampling, all the point cloud are normalized: the point cloud is scaled in order to be contained in a sphere with unitary diameter. The diameter of the point cloud is commonly defined as the measure of the maximum distance between two points contained in the point cloud. This operation is necessary because different point cloud 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 cloud with different scale and diameter. A further explanation about the application of the noise is reported in section 3.3.

The last step of the pre-processing consist in the patch division of the data and it is slightly different for the training, validation set and the testing set. As explained, the network realized performs operation upon points and their neighborhood, therefore the patch has to be created accordingly. From all the files containing the sampled and normalized point clouds few points and their 1024 closer points are selected in order to create a single patch, the central points are chosen sufficiently spaced to obtain diverse patches; the final dataset for the training phase is composed by 117.000 different patches and the validation set by 100, a little selection from the original dataset addressed for the training.

## 3.3 White Noise Application

The point clouds derived from the dataset described in section 3.2 are artificially corrupted in order to simulate a noisy data observation. In particular the noise considered in this project is a *additive white noise*, that affects a point cloud changing the position of each points, modifying the value of the three spatial coordinates.

A easy way to emulate this behaviour is create a matrix of the same dimension of the clear input filled with random, uniformly distributed variables between zero and one, multiply this matrix by the value of standard deviation chosen and add the result obtained at the clear input: the outcome would be a noisy version of the clean input.

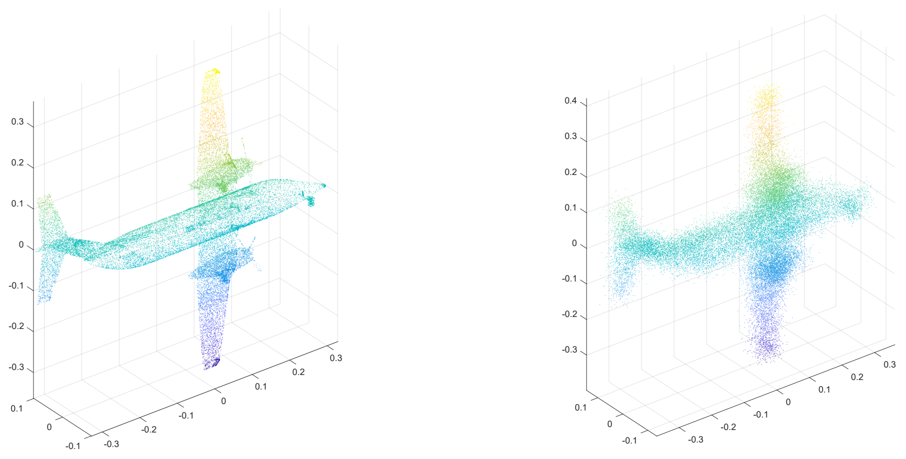
$$\mathbf{X^n} = \mathbf{X} + \mathbf{N} \tag{3.1}$$

$$\mathbf{N} = \mathcal{U}(0,1) * \sigma = \mathcal{N}(0, \sigma^2) \tag{3.2}$$

where $\mathbf{N}$ is the additive white noise with standard deviation $\sigma$ and $\mathbf{X}$ is the original clean input.

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

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



(a) Ground Truth (b) White Noise with $\sigma$=0.02

Figure 3.2: Examples of corrupted data

# Chapter 4

# Graph Neural Network for Point Cloud Denoising

In this section the network proposed is in-depth analyzed and the details of the principal blocks are described.

## 4.1 Architecture design

### 4.1.1 Overview

An overview of the architecture presented, called GraphPointNet, is shown in 4.1. It can be seen that the network is divided into three big blocks: the pre-processing block, two residual blocks and the last part of reconstruction of the denoised point cloud.

The pre-processing block is designed in order to express the noisy observation in the feature space. This different representation is exploited to extrapolate information from the input that will be used in the denoising. During the training, the network progressively learns an efficient description suitable for denoising, able to detect the meaningful information for the required task.

Subsequently the feature extraction, the input is processed by several residual blocks. It has to be recalled that the network exploits the residual learning: the network is trained to learn the white noise instead of directly the cleaned point cloud. Each block is designed in order to detect the residual between the input of the block and desired output, i.e. the discrepancy between the noisy observation and the white noise. The correlations detected in the input are removed at the end of each block; the idea is to gradually eliminate the true point cloud and obtain only white noise.

Finally, the noisy estimation is expressed in space coordinates and subtracted to the noisy point cloud, obtaining the denoised version.

### 4.1.2 Design choice

The network proposed in Fig. 4.1. is generally composed by several graph-convolution layers, followed by batch normalization and Leaky Relu non linear function as most relevant blocks.

The non linear function proposed is a variant of the Relu function described in section 1.4.2: it is equal to the Relu function for positive values but 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 inserted Relu function, if a neuron receives a negative input it would return always zero as output, as shown in equation 1.3, in the case in which many neuron's input become negative for any reason, random initialization or wrong learning rate,the neurons considered are never working.

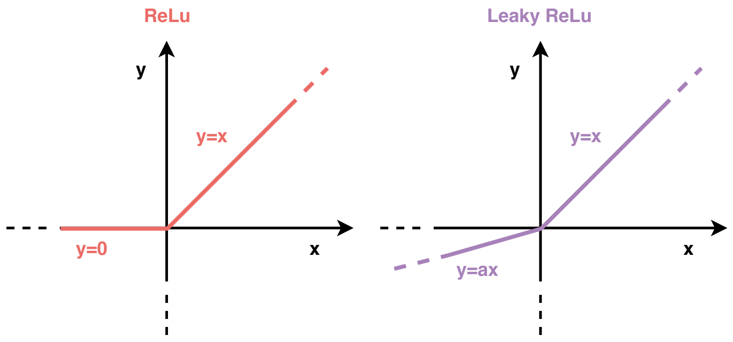


Figure 4.2: Relu vs. LeakyRelu

A noisy point cloud can be represented as:

$$\mathbf{P_n} = \mathbf{P} + \mathbf{n} \tag{4.1}$$

where all the three matrix have the same dimension $nx3$, $n$ number of points in the point cloud.

The network learns to predict the white noise $\mathbf{n}$, that represent the discrepancy between the noisy observation and the clean point cloud. This structural choice is based on work of Zhang et al. [8], 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 normalization 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. The advantages of the residual learning are exposed

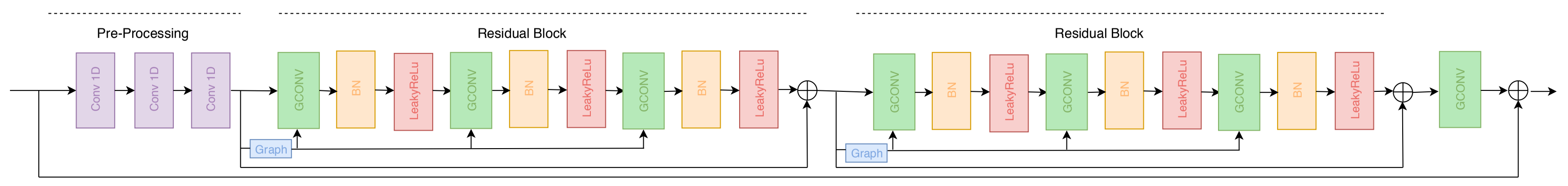


Figure 4.1: GraphPointNet: architecture of the network proposed

in [9], where the performance degradation with increasing depth problem is analyzed. On the contrary to what might be expected, adding layers to an optimized network leads an increment of the training error. Even if a deeper version of an architecture is created with identity mapping as added layers, solvers are not able to find a solution as good as the one associated to the original structure. To overcome this degradation a deep residual network is proposed, that instead of predict the desired mapping $H(x)$, estimated the residual $R(x) = H(x) - x$, where $x$ is the input of the net. Therefore to recover the original function $H(y)$ a inverse operation is applied: $H(x) = R(x) + x$. An example of building block is reported in Fig.4.3.

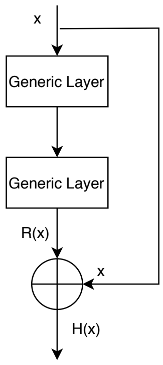


Figure 4.3: ResNet: Building Block of Residual Learning

The reformulation of the problem is developed because the network is not able to well approximate an identity function, assertion based on the fact that the degradation problem is registered even in the case in which identical mapping are inserted. Based on this idea, if the desired function is closer to the identity, exploiting a residual learning can bring advantages. The network learns the discrepancies between the reference and the identity rather than a new function. It can be seen from Fig. 4.3., that the residual formulation $H(x) = R(x) + x$ is realized just inserting a short connection, a simple operation, without increasing the complexity or adding parameter, that turns an original design into its residual version.

In the denoising network proposed several residual connections are inserted; it is shown in Fig.4.1. that besides the connection between the noisy input and the estimates noise in order to recover the denoised point cloud, two internal residual block are designed. Each residual block estimates the noise progressively removing the original point cloud, identifying the discrepancy between noise and the input of each block. A in-depth analysis of a residual block is reported later in the chapter, as well for the a Pre-Processing block shown in figure 4.1.

The novel contribution of the method proposed is the Graph-Convolution layer, that allows the design of a CNN-like structure also for input data that relies on a non regular structure. This type of layer introduces a powerful method for manipulation, extrapolation and representation of complex data, that can be used for several task. This operation implements the equation 1.18, here reported

$$\begin{aligned} H_i^{l+1} &= \frac{1}{|N(i)|}\sigma\Bigg(\sum_{j\in N(i)} F_{w^l}^l(H_j^l - H_i^l)H_j^l + b^l\Bigg) \\ &= \frac{1}{|N(i)|}\sigma\Bigg(\sum_{j\in N(i)} \Theta_{ji}^l H_j^l + b^l\Bigg) \end{aligned}$$

The function $F$ returns a weight matrix $\Theta$, where each elements $(i,j)$is set based on the difference between the feature vector of the point $i$ and the feature vector of a non-local neighboring element $j$. Computing the weight matrix in such way it is possible to assert that the ECC is a general formulation of the classical convolution operation. The function $F$ in the paper [1], is designed as a two layers fully connected neural network, but negatively affects the training with over-parameterization. In order to overcome this problem, a partially structured matrix is exploited for the last layer of the network that implements the $F$ function: multiple row-subsampled circulant matrices are stacked, technique that decrements the number of parameters. Partial circulant matrices with three row for each one has been chosen for the design, constraining the network to have a feature number multiple of three.

The network proposed is finalized to denoising the point cloud and to optimize the parameter of network the *Mean Squared Error* is exploited as loss function:

$$MSE = \frac{1}{N}\sqrt{\sum_{i}^{N}\Big(P_{denoised} - P_{clean}\Big)^2} \tag{4.2}$$

The loss function is computed when the estimated noise is expressed in space coordinates and subtracted to the noisy point cloud.

## 4.2 Block Diagram

### 4.2.1 Residual Block

The simulations are performed instantiating only two residual blocks, due to the high-time consuming operations involved, since each block is composed by several graph-convolution, and utilize a deeper model, in the first stages of the development, would only have slowed down the process.

Each block is composed by three layers of graph-convolution that share the graph, that is computed upon the input of the block itself. Finally, each graph-convolution is followed by a batch-normalization layer and the activation function, in Fig. 4.4. the global structure is shown, with all layers instantiated.

The input data are normally distributed, an important characteristic that helps the network in the training phase, but as deeper is the network this property is easily lost. The batch normalization block is in charge of preserve the normalization of the data through the net. The activation function exploited in the whole project is the Leacky-ReLu function, as reported in the figure 4.4.

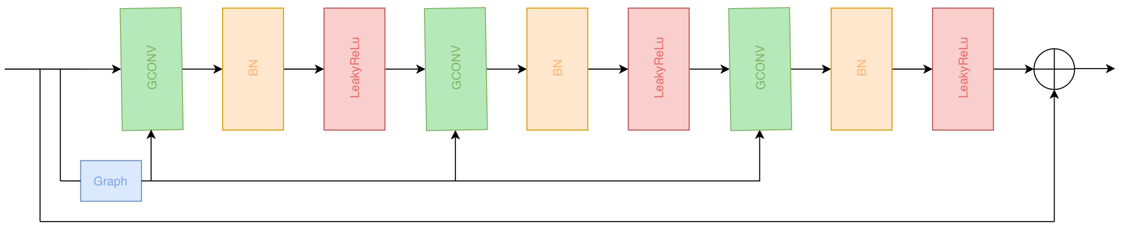


Figure 4.4: Architecture of Residual Block

### 4.2.2 Pre-Processing Block

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

The network based on graph convolution do not take as input the data described in the spatial domain but in the features domain, where, intuitively, more features are involved more information the network is able to detect. On the other hand, enlarge the number of feature causes a non negligible increment in the data stored and reached a specific performance, the network is not able to detect more information even after further expansions. Moreover, it has to be recalled 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 occupancy of the memory. In early simulations the number of features chosen is 66, successively enlarged to 120.

It is important to discuss the implementation of the feature extraction: for this purpose, a 1-D convolution is exploited. This operation can be seen as a multiplication for a matrix with dimensions set properly for the number of feature desired, followed by ad addition for a bias. It has be seen empirically that gradually enlarging the features number in a deep structure achieves better results than exploit just one layer net. In Fig. 4.5. it is possible to observe the implementation to obtain 66 features. The batch normalization layers are inserted in order to obtain data consistent with the residual block: as shown in Fig. 4.1. the output of the pre-processing and the first residual block are combined.

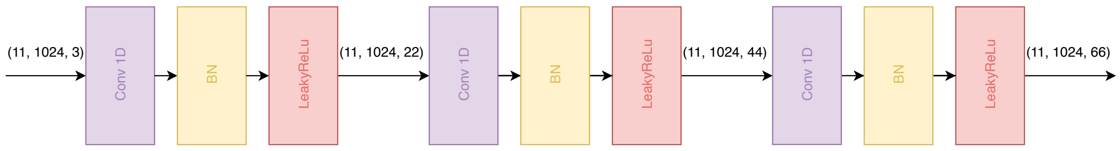


Figure 4.5: Architecture of Pre-Processing Block

### 4.2.3 Graph Convolution layer

The *Graph Convolution layer* is the core of the network: it takes as input a feature vector $\mathbf{H}^l$ and compute a new feature vector $H_i^{l+1}$ associated to each point $i$, obtained from the weighted local aggregation of the feature vectors $H_j^l$, for $j \in N_i^l$, where $N_i^l$ is the neighborhood of the point $i$ at layer $l$. The adjective *local* used in the description of the operation actually means *local in the feature space.* The peculiarity of the graph convolution operation is to consider close into the feature space not only the points that are *spatially nearby*, but also points that *share similarities.*

In order to detect the points $j$ belonging to the neighborhood of each point $i$, it is necessary to build the edge-labeled graph of the point cloud, where the edges between each pairs of nodes are estimated as the difference between the feature vectors associated to them. After the distances computation, the neighborhood of each node is defined: for each node $i$ a fixed number of closer points is selected. Different numbers of neighbors are tested: early simulations are characterized by eight closer points considered and then the value is increased, reflecting an increment of the network global performance.

In Fig. 4.6. the implementation of the operation described above is reported. First of all, it is possible to observe a block called *Graph*, outside the main element *Gconv*: in this block the distances between all the points in terms of Square Euclidean Distance are computed, allowing to build the graph, necessary for the graph convolution operation. Inside the fundamental element it is reported a block called *Non-local/local aggregation* where the operation of ECC [1], the weighted aggregation, is implemented for each point $j$ in the neighborhood of the center point $i$. Moreover, an additional block of 1-dimension convolution is exploited to take into account the self loop,i.e. the center point $i$ of each neighborhood. In order to obtain the new feature vectors the mean of the two components over described is considered.

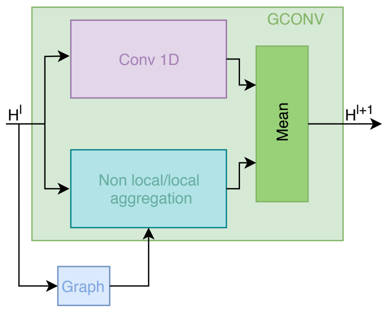


Figure 4.6: Architecture of Graph Convolution Layer: $H^l$ is the input of the layer $l$ and $H^{l+1}$ is the output of the graph convolution layer that will be the input of the $l+1$ layer

# Chapter 5

# Validation and results

## 5.1 Testing

Once the network is trained, the testing is performed, in order to evaluate the performances, the goodness of the denoising results and make comparisons with other methods.

The dataset of the training phase is composed by non-overlapped patches extrapolated from a point cloud: a point, that will be the center of the patch, is selected and its 1024 nearest points are collected, creating a patch.

During the test phase the convolutional structure of the training data has to be replicated properly, but whole 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 jointly all the points of the point cloud. To perform properly the convolution operation over the whole point cloud a specific code for the testing net is edited. All the non-convolutional operations are reported with no difference from the description in the training code and are directly computed over the whole point cloud. Instead the convolutional operations are executed just over one point by time considering its nearest neighbors, extracted from the *nearest-neighbors matrix*: the convolution operation takes as input one point and its neighborhood and returns as output just the modified point. This operation is executed in parallel and the whole point cloud after a *graph convolution* layer is collected and carried on in the network.

Once the denoised point cloud is obtained, the *Cloud-to-Cloud distance*, also called *C2C*, is exploited to evaluate the goodness of the method. This metric is adopted in several projects concerning the point cloud denoising, such as [3] and [7], but occasionally called in different ways despite the same computation. It consists in measuring the mean distance between the points of the denoised point cloud and the ground truth, i.e. the original clean point cloud. More in detail, it is computed calculating the mean of the square root of the squared Euclidean distances between each point of the ground truth and its closest denoised point, then measuring the opposite distances, between each denoised point and its closer point in the ground truth point cloud and evaluate the average as well; the final *C2C distance* is the mean between the two distance estimations.

$$C2C = \frac{1}{2}\Big[\frac{1}{N_{gt}} \sum_{gt_i \in G_t} \sqrt{\min_{dn_j \in D_n} ||gt_i - dn_j||_2^2} + \frac{1}{N_{dn}} \sum_{dn_i \in D_n} \sqrt{\min_{gt_j \in G_t} ||dn_i - gt_j||_2^2}\Big] \tag{5.1}$$

where $G_t$ is the collection of the points of the ground truth, $D_n$ of the points of the denoised point cloud, $N_gt$ and $N_dn$ the total number of points in the sets respectively.

The testing-set is composed by ten different point cloud for ten categories of the data-set Shapenet [11]; the data are pre-processed as discussed in section 3.2. In the tables below are reported the results of several simulations, exploiting different feature numbers and applying distinct level of noise in terms of standard deviation.

For each simulation one table for category is reported, where the results for each of the ten models selected are presented; the Shapenet [11] dataset contains a large amount of point cloud for each category, therefore the number of each point cloud is shown in the left column for clearness.

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

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

The authors of the project based on graph Laplacian regularization [7], called *GLR* kindly provided the MATLAB code that is tested over the same testing set of GraphPointNet, in order to have comparable results.
The project PointCleanNet is available on *github*, and a pre-trained model for a blind denoising is available on the portal. The network is tested and the results are reported for comparisons.

Following the C2C distances computed from the denoised version of Graph-PointNet, GLR and PointCleanNet are reported.

### 5.1.1 Simulations with Noisy Point Cloud with standard deviation equal to 0.02

The first test performed considers GraphPointNet trained for 900.000 iterates, with feature number equal to 66. The testset is the same for all three methods and it is affected by white noise with standard deviation 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 5.1: Airplane 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 5.2: Bench 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 5.3: Car 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 5.4: Chair 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 5.5: Lamp 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 5.6: Pillow 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 5.7: Rifle 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 5.8: Sofa 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 5.9: Speaker 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 5.10: Table testset corrupted by 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** |

Table 5.11: Category Mean for simulation for white noise with $\sigma = 0.02$

In the table 5.11 the mean over all the models of each category is shown, in order to easily understand which method achieves on average the better results. It can be seen that GraphPointNet do not achieves the best results only for the Rifle category. Regarding this category, the graph-network proposes a denoised version of the point cloud slightly worse than the one denoised by PointCleanNet network; however it is able to outperforms the GLR method's results.

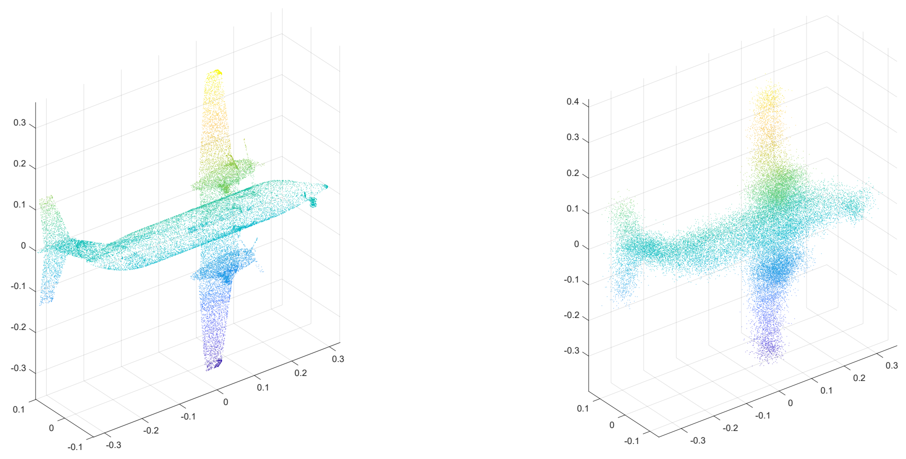
| Category | 66 Features | 120 Features |
|---|---|---|
| Airplane | 0.005491 | 0.005434 |
| Bench | 0.005422 | 0.005346 |
| Car | 0.006774 | 0.006783 |
| Chair | 0.006691 | 0.006548 |
| Lamp | 0.005353 | 0.005105 |
| Pillow | 0.005407 | 0.005210 |
| Rifle | 0.005855 | 0.005535 |
| Sofa | 0.007039 |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

Table 5.12: Comparison between GraphPointNet with 66 and 120 learning features

A possible improvement of the network is to increase the number of features, in order to achieve better performance.
It is interesting to analyze the behaviour of the network in this new scenario: a network with the same architecture presented in the previous chapters but with a feature number incremented from 66 to 120, is trained and tested. A comparison between the different nets is at the same level of standard deviation considered

for the noisy input is analyzed and reported in 5.12.
Expanding the number of feature, the network is able to extract more information from the input data, therefore there should be registered an increasing of the performance. It can be seen from table 5.12 that an improvement on average is reveled. Only in two categories can be noticed a slightly degradation of the performance; even with this degeneration, in the two categories affected the GraphPointNet still achieves the best performance.
A more relevant observation regards the Rifle category. As shown in table 5.11, the Rifle category is the only one in which GraphPointNet do not achieves the best performance, but increasing the number of feature, it would provide the best denoised point cloud in terms of c2c metric, beating the PointCleanNet network.

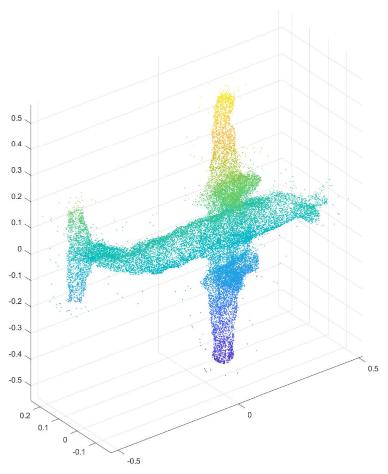
Furthermore, a qualitative comparison between the denoising methods described is presented. In Fig. 5.1. the original point cloud and the noisy version, corrupted by white noise with standard deviation equal to 0.02, are reported and in Fig. 5.2. the different denoised version of the airplane are shown.



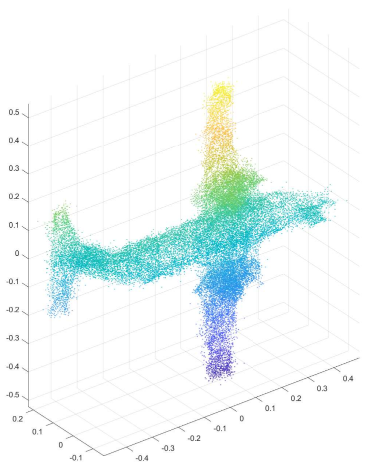
(a) Ground Truth (b) White Noise with $\sigma$=0.02

Figure 5.1: Airplane model: Original and Corrupted Point Cloud

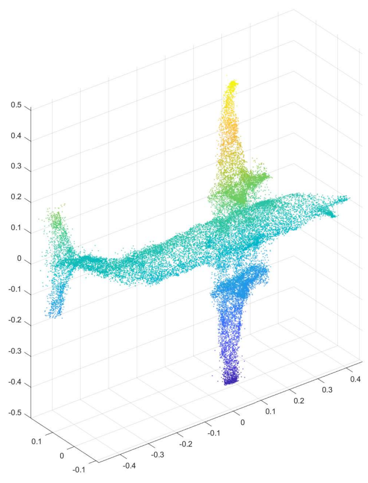
It can be visible from the denoised point cloud produced by PointCleanNet that this method is not able to efficiently project all the points upon the point cloud surfaces, leaving a lot of points as outlier. Instead, the GLR algorithm is not able to recover the detail of the global shape of the point cloud: the denoised version remains wider than the original one. GraphPointNet, the method proposed, is able to sufficiently reconstruct the shape of the original point cloud, with fewer outliers points. Some details of the original point cloud are visible in point cloud in Fig. 5.2c., as the front of the plane.



(a) PointCleanNet



(b) GLR



(c) GraphPointNet

Figure 5.2: Airplane model: Denoised Point Cloud results of the methods analyzed

### 5.1.2 Simulations with Noisy Point Cloud with standard deviation equal to 0.01

Here is reported the test with point cloud affected by white noise with standard deviation equal to 0.01 The second test performed considers GraphPointNet trained for 900.000 iterates, with feature number equal to 66, as in the previous simulation reported, but the testset is affected by white noise with standard deviation 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 5.13: Airplane 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 5.14: Bench 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 5.15: Car 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 5.16: Chair 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 5.17: Lamp 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 5.18: Pillow 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 5.19: Rifle 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 5.20: Sofa 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 5.21: Speaker 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 5.22: Table testset corrupted by 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 |

Table 5.23: Category Mean for simulation for white noise with $\sigma = 0.01$

As previously, the table 5.24 shown the mean per category for an easier comparison between the methods analyzed.

| 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 |

Table 5.24: Category Mean for simulation for white noise with $\sigma = 0.01$

It can be visible that with a lower white noise the method GLR obtains promising results, with performance similar to the method proposed. This optimization method achieves very good result if the point cloud is corrupted with a low noise, otherwise, as shown in the previous experiment, it is not able to provide a good denoised point cloud able to compete with the results of the other method considered.
It can be notice that GraphPointNet is able to produce a denoised point cloud very close to the state-of-arts and outperforms the other neural network taken into account for the comparison, PointCleanNet, that achieves the worst results but for a category.

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

It is visible from the simulations reported that at high level of noise GraphPointNet achieves the best results overcoming all the other methods. Instead, considering a lower level of noise it still provides a good denoised version of the point cloud, comparable to the best results provided by the state-of-arts, but the gain margins are reduced.

# Chapter 6

# Conclusions and further developments

In this thesis an innovative neural network designed for denoising point cloud is presented.

As discussed in previous chapters, the novelty lies on the introduction of the graph-convolution layers, that exploit the Edge Conditioned Convolution, a general convolution formulation.

It is verified from the simulations presented that the network is able to outperform the current stae-of-art. It has been shown that the most promising result are achieved when the original point cloud is corrupted by a high level of noise, as presented in table 5.11 and more in detail in tables for category, where the network provides the best denoised version for most of the categories selected for the testing-set. Considering point cloud characterized by a lower additive white noise, the GLR method became more effective and the performance achieved by this method and the one proposed are similar, as shown in table 5.24. It can be notice that even in this scenario, the method proposed obtain better performance in the majority of the category.

## 6.1 Future works

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

As reported in chapter 4.2.3, during the training the network learns to improve the denoised point cloud provided just having information about the closeness of the estimated points with the ground-truth one. It is not necessary to fully recover the exact position of each point of the point cloud, it is important to move the noisy points equally distributed over the original surfaces of the point cloud.

Therefore, it could be useful enlarge the network and insert a block specialized in the surface estimation for each point.

As a future improvement of GraphPointNet is proposed to modify the loss function, introducing a term that takes into account the normal of the points at the surface. In particular, a possible development is to estimate the normal at the surface for each of the denoised point and compare it with the ground-true normal, and force the network to decrease the discrepancy.
A new loss function is created, the norm of the difference between the norm estimated and the original is appended to the original loss, $MSE$: this improvement would produce a network able to better estimate the denoised point cloud avoiding outliers and leading to an increase of the performances.

# Glossary

**Convolutional Neural Network** Class of Deep Neural Network with wide application in Image Processing.

**Deep Neural Network** .

**Denoising** Procedure finalized to recover from an input corrupted by any type of noise to its original version. It is a typical task of Image Processing.

**Edge Conditioned Convolution** Generalization of the classical convolution, based on a graph representation fo the data, presented in [1].

**GraphPointNet** The network proposed in this thesis that addresses the task of point cloud denoising exploiting a graph-convolutional architecture.

**Mean Squared Error** Formula that measures the discrepancy between the estimated value and the original one. It computes the average of the squares of the difference between the inputs.

**Neural Networks** Algorithms, ispired by the human brain, able to recognize patterns.

**Point Cloud** It is a 3-D object representation.

**Preprocessing** Building block of the network in charge of the represent the input in the feature space. In GraphPointNet the Preprocessing block move the noisy point cloud, expressed in the 3-D space to a representation over 66, or 120, features.

**Residual Block** Building block of a network characterized by residual learning. The block is in charge of estimate the discrepancy, the residual, between the expected value and the input. In GraphPointNet each residual block eliminate the correlation from the input of the block, ideally producing in output white noise.

**Residual Learning** Method exploited in neural network, the network is trained to learn the residual between the expected value and the input rather than directly the desired estimation.

# Bibliography

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

[2] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[3] Marie-Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, Niloy J Mitra, and Maks Ovsjanikov. Pointcleannet: Learning to denoise and remove outliers from dense point clouds. *Computer Graphics Forum*, 2019.

[4] Jelena Kovacevic Chaojing Duan, Siheng Chen. 3d point cloud denoising via deep neural network based local surface estimation. *Computer Vision and Pattern Recognition*, 2019.

[5] Rémi Gribonval David K. Hammond, Pierre Vandergheynst. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis, Elsevier*, 30(2):p.129–150, 2011.

[6] Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. PCPNet: Learning local shape properties from raw point clouds. *Computer Graphics Forum*, 37(2):75–85, 2018.

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

[8] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[10] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. *Proceedings of 28th IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[11] Hao Su, Jianxiong Xiao, Li Yi, Fisher Yu, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Fisher Yu, Qixing Huang, Qixing Huang, Zimo Li,

Silvio Savarese, Manolis Savva, Shuran Song,. ShapeNet: An Information-Rich 3D Model Repository. *Proceedings of 28th IEEE Conference on Computer Vision and Pattern Recognition*, December 2015.