Neural Networks and Deep Learning

Final Project: 3D Object Classification Using Point Clouds and Graph Convolutional Networks

Gianpietro Nicoletti[1], Matteo Candon[2], and Nicola Rizzetto[3]

[1]Department of Information Engineering, ICT for Internet and Multimedia,University of Padua
[2]Department of Information Engineering, ICT for Internet and Multimedia, University of Padua
[3]Department of Information Engineering, Computer Engineering, University of Padua

***Abstract*—With the recent widespread of immersive technologies, 3D data formats (e.g. meshes and point clouds) are becoming every day more important. In order to manage them in a fast and efficient way, Neural Networks and Deep Learning approaches come in handy. The scope of this project is to develop a Neural Network able to solve the 3D Model Classification task, based mainly on Graph Convolutional filters. For this reason, ModelNet10 has been chosen as the practice field: it is a dataset of 3D objects divided in 10 different classes, and provides a total of 4,899 pre-aligned shapes. In order to confront the results with an existent and state-of-the-art network approach, the PointNet model has been taken into consideration. It is a model that is able to perform 3D Model Classification, Part Segmentation and Semantic Segmentation on point clouds. In the following chapters will be presented the pipelines of both models, the dataset creation (since the dataset objects are given as meshes, they have been first converted to point clouds before feeding them to the two networks) and the obtained results from the training sessions.**

***Index Terms*—Supervised Learning, Neural Networks, Graph Convolutional Networks, Point cloud, Classification, Computer Vision.**

I. Introduction

Point clouds are one of the most used way to represent 3D objects in the field of Computer Vision. This is due to the fact that various devices producing 3D point clouds have become widely applicable in recent years, e.g. range sensors in cars and robots or depth cameras like the Kinect [1]. The advantages of using this representation are connected to the huge amount of devices that allow the acquisition of 3D points coordinates with high accuracy and details, to the simplicity itself and the possibility to be used in real-time applications [2]. In both supervised and unsupervised learning, these point clouds have many interpretations, varying from tridimensional matrices for Convolutional Networks to undirected graphs for Graph Convolutional Networks or GraphSAGE.
In this paper, the classification of objects represented through point clouds is investigated using Modelnet10 as dataset and graph convolution as main model. The results using the same dataset are then compared with the PointNet model [3].

II. Related Work

A. Classification strategies

The main problem with point clouds is the sparcity, caused by the tecnology used to collect (e.g. LIDAR), the unordered and unstructured representation of the data that does not allow to process them in some easy way using basic neural networks [4].
Some of the main techniques to process point clouds are based on:

- Structured grid: such as voxel, that allows to represent the point in some volumetric structure, or a 2D representation based on projection of the points into a 2D image and processing them using a standard convolutional model;
- Raw point cloud: used for example in networks like PointNet without giving them a structure, or converting the representation into a graph, giving them a structure but mantaining the information of the original points.

Among all of the alternatives presented in literature, the project is focused on structuring and processing the raw point clouds, considering the PointNet network as reference. Summarizing the concept, PointNet extracts the object features processing each point separately and then aggregates them into a global features that represents the object [3].

B. Graph Convolutional Networks

To solve the problem of expressing the similarity between nodes of a graph, a compression system was developed, named node embedding [5]. A node embedding system is composed of:

- Similarity: function that takes two nodes u and v from the graph and returns their similarity (can be the adjacency matrix value relative to (u,v) or the Jaccard similarity);
- Encoder: function that takes a node v and maps it into a $\mathbb{R}^d$ space as an embedding z_v;
- Decoder: function that takes two embeddings z_v, z_u and computes their similarity;
- Loss: function that measures the discrepancy between the real similarity and the similarity of the embeddings of nodes u and v.

The most used node embedding system is shallow embedding. Explaining it briefly, it takes a matrix Z ($d \times n$) of embeddings and given a node v it returns the corresponding embedding z_v ($z_v = Z * v$, with v vector of all zeros except a 1 representing the node v). To learn the most appropriate embedding, the encoder has learnable parameters, but this system only learns the embeddings of the training nodes, so it can not create embeddings for new nodes. In addition, shallow embeddings can not use the features of the nodes, so a Neural Network

framework for graphs has been implemented. A GNN is essentially a node embedding system but with a Neural Network as encoder, so it maintains the rest of the structure. This augmented system can also be used for Node Classification, Link Prediction or Graph Classification, as done in this paper. A GNN takes as input a vector x_v = $[A_v, F_v]$, where A_v is the adjacency vector relative to node v and F_v is the vector of node features of v. At every layer the GNN applies two functions:

- AGGREGATE($h^{(k)}(u)$, $\forall u \in N(v)$): function that aggregates the information from the embedding of the neighbour nodes into one message $m^{(k)}_{N(v)}(v)$:

$$AGGREGATE(h^{(k)}(u), \forall u \in N(v)) = \sum_{u \in N(v)} h^{(k)}(u) = m^{(k)}_{N(v)}(v)$$

- UPDATE($h^{(k)}(v)$,$m^{(k)}_{N(v)}(v)$): function that takes the information from the neighbour message and the embedding of node v to create a new embedding (σ = activation function, $b^{(k)}(v)$ = bias relative to node v, $W^{(k)}_{self}$ = matrix of the weights for $h^{(k)}(v)$, $W^{(k)}_{neigh}$ = matrix of the weights for the neighbourhood of $h^{(k)}(v)$):

$$UPDATE(h^{(k)}(v), m^{(k)}_{N(v)}(v)) = \sigma(W^{(k)}_{self} * h^{(k)}(v) + W^{(k)}_{neigh} * m^{(k)}_{N(v)}(v) + b^{(k)}(v))$$

Graph Convolutional Networks are a variant of Graph Neural Networks focused on generalization [6]. The difference in the structure of the GCN comes from the integration of node v in the aggregation and normalizing the aggregated information.
Integrating node v is a form of self loop, which helps avoiding overfitting but reduces the expressivity of the network. Since the embedding of v is processed inside AGGREGATE() the matrices of the weights become $W^{(k)}_{self}$ = $W^{(k)}_{neigh}$ = $W^{(k)}$.
Normalizing the aggregated information helps having a numerically stable model, since nodes with a higher degree will have a higher learning ability than the nodes with low degree, thus creating imbalance (intuitive result). Normalization can be done considering only the degree of v or both v and the considered node at the time, as done in the GCN. Considering these changes, the functions become:

$$AGGREGATE(h^{(k)}(u), \forall u \in N(v) \cup v) = \sum_{u \in N(v) \cup v} \frac{h^{(k)}(u)}{\sqrt{(|N(v)| * |N(u)|}} = m^{(k)}(v)$$

$$UPDATE(m^{(k)}(v)) = \sigma(W^{(k)} * m^{(k)}(v))$$

This can be summarized by writing the propagation rule for the layer, in the matrix form:

$$\sigma(D^{-\frac{1}{2}} \hat{A} D^{-\frac{1}{2}} F W)$$

Where D and $\hat{A}$ are the degree and the modified adjacency matrix respectively ($\hat{A} = A + I$, that adds the self loop to each node) $n \times n$, F is the features matrix $n \times i$, where i is the input dimension of the features space, W is the weight matrix $i \times o$ where o is the output dimension of the features space.

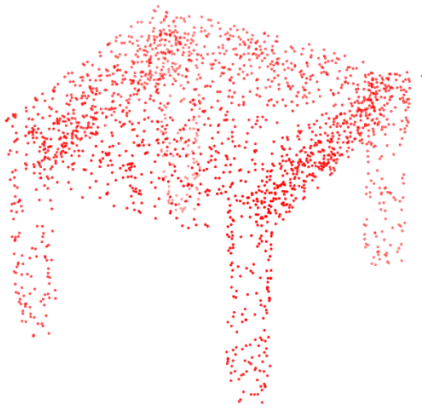


Fig. 1: Example of an input point cloud from ModelNet10.

III. Process Pipeline

In this section are reported the main steps of the process, with the intention of explaining some of the choices made and provide a summary of the entire work.

A. Data extraction and representation

As mentioned in section II, one of the main problems is the lack of structure inside the point clouds. So, in order deal with this fact, a graph rappresentation is provided. The points were sampled from the CAD models present in the ModelNet10 dataset, of which an already sampled example is reported in Figure 1.

B. Definition of the model

A Deep Learning technique must be defined to extract the features from the points in order to achieve the goals of this project. In particular, based on the same idea of PointNet, a feature extraction from the points and their aggregation was adopted. The results on the same dataset were then tested and compared with the original network.
Together with the model definition, some improvements to stabilize the training procedure and increase the accuracy must be considered.

IV. Signals and Features

Given the CAD models in ModelNet10, a random sampling of 2048 points was performed for each object in order to build the point clouds. With this technique, only the information about the position of each point was retrieved, which is also the only information used in PointNet. In order to increase the features of the points, an estimation of the normals was performed using the implementation provided by the Open3D library for Python. An example is provided in Figure 2.

With this unstructured set of points, a KNN-graph was build considering for each point the 50 nearest neighbors. This allowed to build from each point cloud a graph with 2048 nodes, containing the features (i.e. position and normal), and 102400 edges.

Fig. 2: Example of an input point cloud including the estimated normal vectors.

Considering that ModelNet10 is already split into a training set with 3991 samples and a test set of 908 samples, more samples were added for the training part by applying some noise to the original data. In particular 11973 new objects were obtained, divided into: 3991 samples that are the original data, 3991 samples obtained adding gaussian noise with mean 0 and variance 5, 3991 samples obtained adding gaussian noise with mean 0 and variance 10.
This choice could appear strange, but the idea behind is to give the model some flexibility considering part of the samples with "small" noise and another part with "high" noise. To be noted also that the point clouds obtained from the sampling are not normalized, so such high variances are not as destabilizing as it may seem. In particular, the idea is that the model tries to adapt to something between the training data and something totally different from it, which can lead to less overfitting.
Lastly, the training set was also split with the purpose of introducing the validation set (10500 samples for training, 1473 for validation).
Regarding the labels, a simple mapping was performed, i.e. a scalar value in range [0,9] for each class.

V. Learning Framework

In this section, the model definition and the training procedure are reported. In particular, the model was designed following the main idea of PointNet, but with the addition of the capability to investigate the structure given by the graph built from each point cloud.

A. Model definition

In Figure 3 the block-diagram representation the model is shown with the main subblocks that compose it:

- **Input**: in the input layers, the model receives a single point cloud divided into nodes features and edges of the graph representation. The features are divided into position features (i.e. (X, Y, Z) coordinates) and normals features (i.e. (N_x, N_y, N_z), estimated normal vector of each points).
- **Points Features processing:** divided in:
  - Two consecutive Graph Convolutional layers used to extract the features of each point from the connected nodes.
  - A multilayer perceptron used to compress the features into a lower dimensional space.
- **Object features processing**: after the convolution and compression, positional and normal features of each point are concatenated, in order to form the object features that will be used to classify the object through a final multilayer perceptron.

B. Training procedure

For the training phase, the labels were redefined using a similar idea of the one-hot encoding, with the difference that instead of giving 1 as value, a weight w was given in the corresponding index of the label. This choice was made to force the network to predict a weight that is able to represent the probability of being in that specific class for the considered sample, where simple SoftMax and logSoftMax functions have failed with traditional one-hot encoding. Morover the rescaling of the loss can speed-up the convergence of the model [7]. The training was done using the MSE loss function, which together with the modified one-hot encoding explains why the losses are that high. Moreover the scaling on the data was not applied.
The choice of the MSE loss may appear strange, but it was empirically proven that MSE may work better on certain models [7].
In particular, thanks to empirical observation, $w = 1000$ was found to be a good value for the weight. In order to balance the bigger difference inside the input label, the updates were performed considering 20 inputs each time, accumulating their gradients and compute the mean to reduce the noise caused the difference of the various inputs. Moreover, to converge as close as possible to the global minima, a decreasing learning rate was adopted: a descent by a factor 10 every 10 epochs.
During the training, which took 30 epochs, 2 different models were saved: one with the lowest training error and one with the lowest validation error. The trends of the losses together with the epochs when the models were saved are reported in Figure 4 and Figure 5.

To better justify the choice of the loss and why the data was not scaled, in Figure 6 and 7 are provided two experiments done on the model:

- In Figure 6 the training procedure of the network using the classical combination of standard one-hot-encoding, SoftMax as output activation funtion and Categorical Cross Entropy as loss.

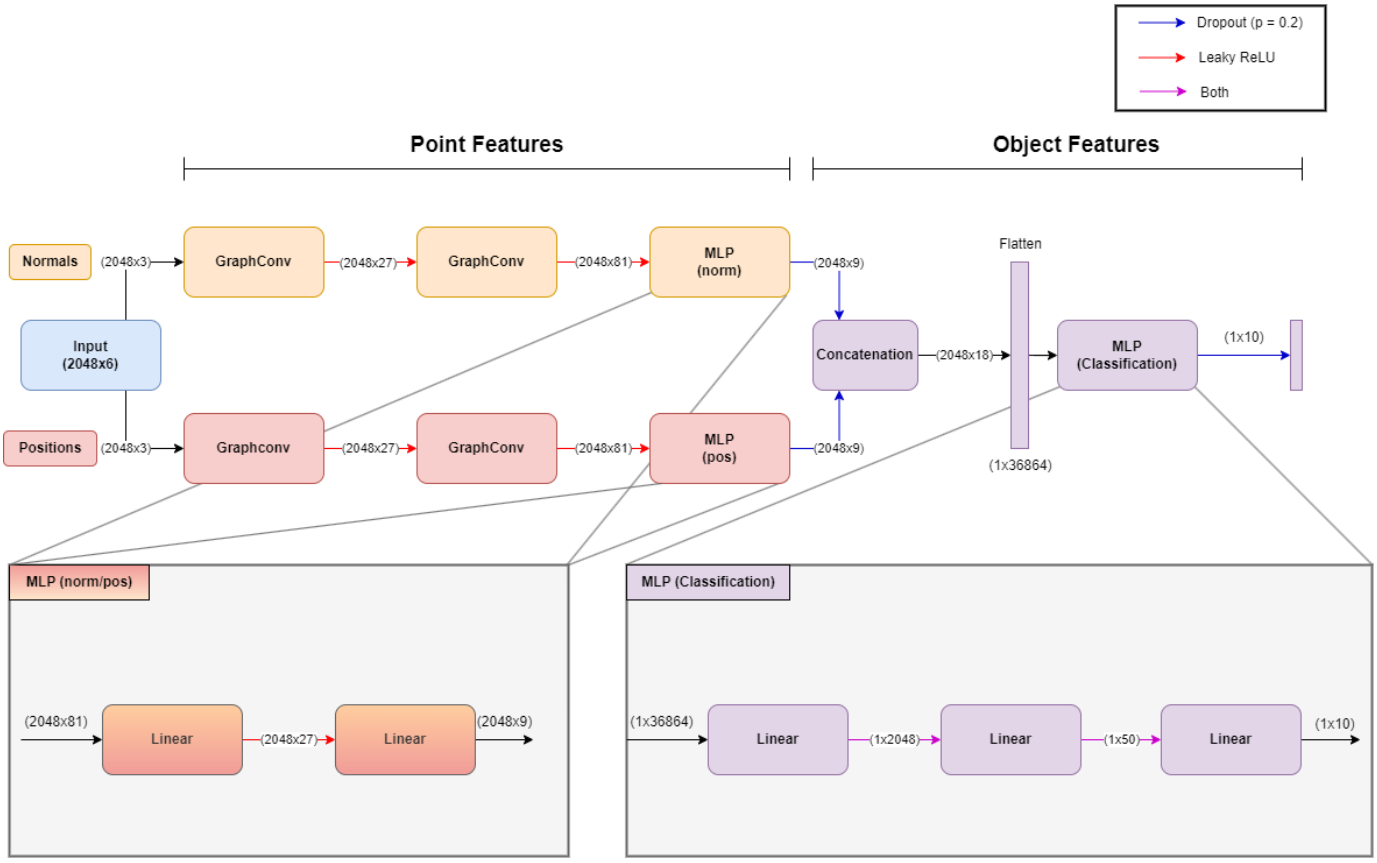


Fig. 3: Design of the model.

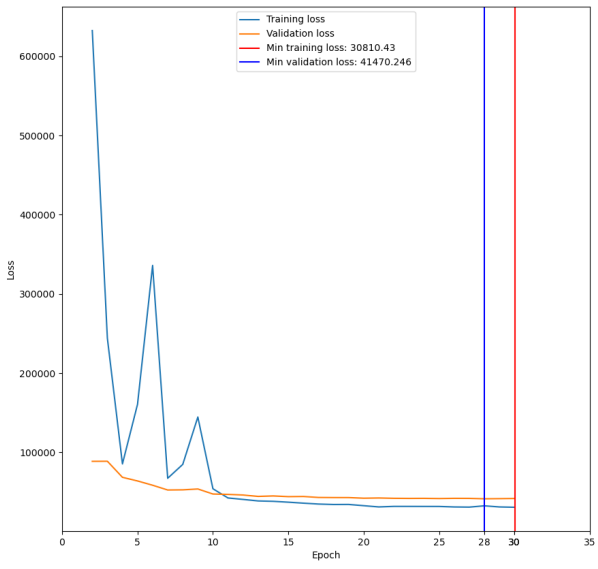


Fig. 4: Trends of train and validation loss during the traning of the network.

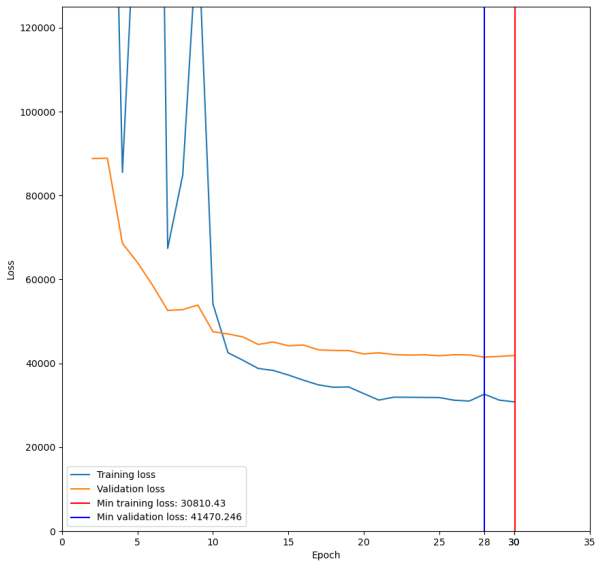


Fig. 5: Trends of train and validation loss during the traning of the network.(zoomed)

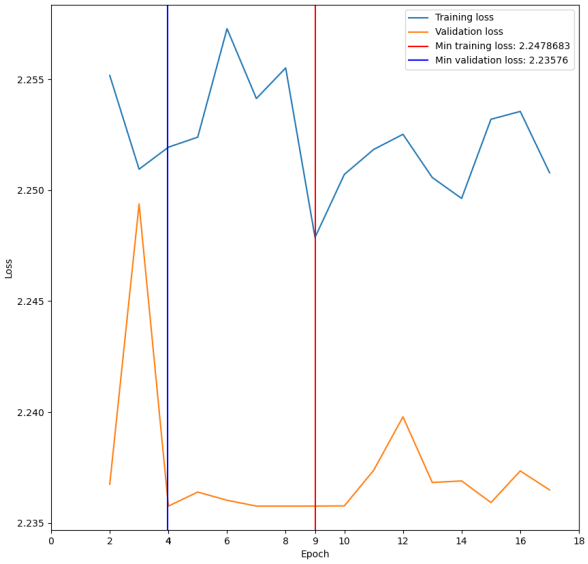


Fig. 6: Trends of train and validation loss during the traning of the network using the Cross Entropy loss and the scaling of the data

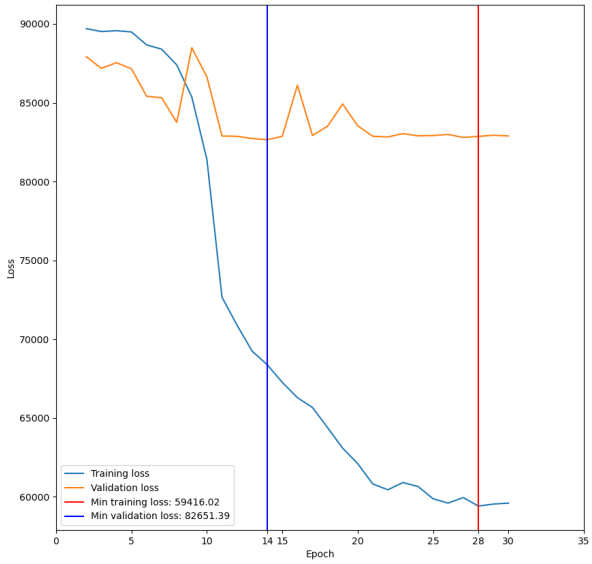


Fig. 7: Trends of train and validation loss during the traning of the network using the MSE loss and the scaling of the data.

- In Figure 7 the training procedure of the network using the MSE loss.

In both cases a min-max normalization scaling was applied. As reported, in both cases the validation loss does not converge. In particular, for the model with the Cross Entropy loss the best accuracies were 22.28% and 11.01% for the train and test dataset respectively, whereas in the model with the MSE loss the best accuracies achieved were 23.72% and 32.93% for the train and test dataset respectively.
Without going further in the details, in general dealing with normalization for structured data required some advance normalization tecnique in order to avoid loss of structural information [8].

VI. PointNet model

In this chapter will be presented more in detail the pipeline of the PointNet model, as presented in the literature [3]. After that, some considerations are presented, such as differences in the dataset w.r.t. the inputs of the other model and some modifications to the network itself.

A. Model description and pipeline

Unordered structures used for 3D Classification like point clouds are often reorganized as 3D Voxel Grids or sets of images, but in this way the data increases significantly in volume and is subject to quantization. PointNet is a deep learning framework designed to efficiently consume unordered 3D point sets, studied to be a unified approach that can work with Model Classification, Part Segmentation and Semantic Segmentation. For this project, only the first task is taken into consideration.

Since the input points are unordered, the network needs to be invariant to permutations. In addition to this, the learned representation should be invariant to certain transformations, such as rotation and translation (if applied globally to the set of points). In the paper experiment, the sampling of the objects returned 1024 points per cloud and the coordinates are normalized in a unit sphere. In order to make the model more robust, a data augmentation process was performed on the dataset: the objects were randomly rotated and the position of each point was jittered by a Gaussian noise (0 mean, 0.02 standard deviation).

The network architecture, as can be seen in Figure 8, is composed by two main modules: a classification network, which takes as input the point clouds, and a segmentation network, which provides a concatenation of global and local features, giving outputs per point scores. Since the task for this project is only the classification of 3D objects, the second network was not implemented and will not be discussed.

The network used has 3 main blocks:

- joint alignment block, or T-Net: the first time, it transforms the input features (n,3) into a canonical representation, while the second time it provides an affine transformation for alignment in the feature space. This transformation is set to be similar to an orthogonal matrix;
- symmetric block: the input gets approximated by a multilayer perceptron (*mlp* in the schematic),then a max pooling layer extracts and aggregates the information from all the points;

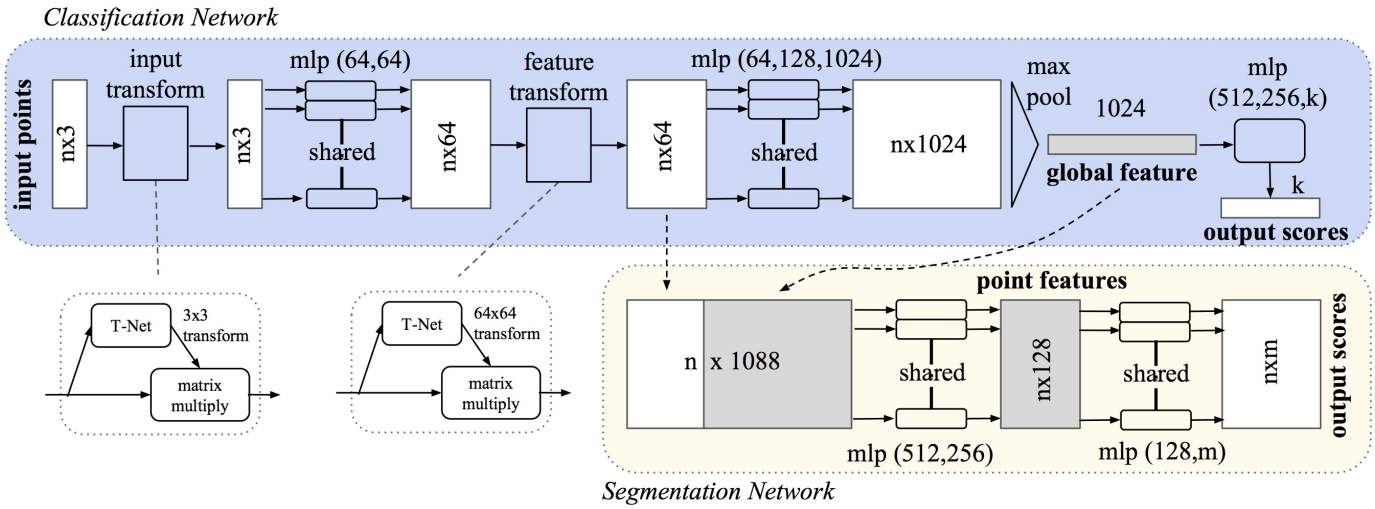


Fig. 8: Design of the PointNet model.

- local and global information aggregation block: the output from the symmetric layer is taken and each global feature is concatenated with the features of the single points, obtaining both local and global information. Then new features are extracted from the concatenated vectors.

The results obtained from the network are comparable with other state-of-the-art approaches, but the PointNet is much faster than the others (to name some of the other models taken into consideration, *MVCNN*, *3DShapeNets*).

B. Considerations

Some aspects of the original networks were changed in order to apply it to this project since the purpose is to compare Pointnet with the developed model. The differences are discussed in the following sections.

C. Training

As said above, for a clear confrontation of the two networks a training of the PointNet was also performed. This is due to the fact that the dataset built was fed to both networks, and the training process was done over the same enviroments. So, the dataset with the same gaussian noise was used. Figure 9 represents the trends of training and validation loss during the main session of training, lasted around 30 epochs.

Regarding the training sessions, some implementations were applied:

- **RMSprop**: the optimizer chosen is the RMSprop, and were used the same parameters of the developed network;
- **Learning Rate Decay**: the learning rate of the optimizer is reduced by a factor 10 every 10 epochs.

The only difference w.r.t. the developed model during the training was the loss and the encoding of the label: Categorical Cross Entropy and standard one-hot encoding.

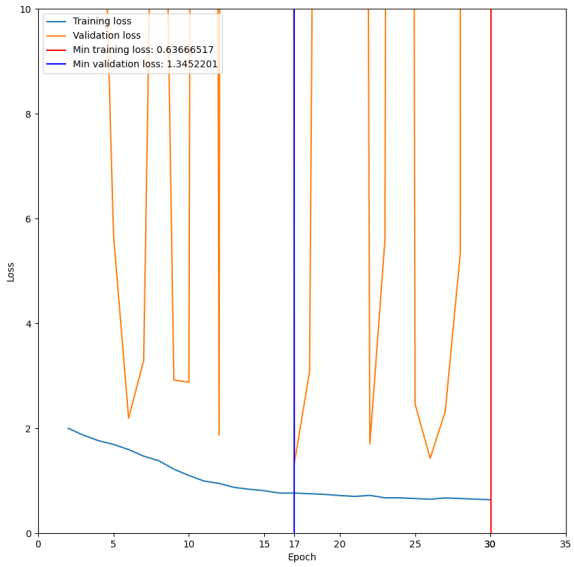


Fig. 9: Trends of train and validation loss during the PointNet model traning.

VII. Results

A. Developed Model

The purpose of this project was to confront the best model between the two obtained in the training sessions (referring to the best training and best validation models) by evaluating their performances on the dataset chosen. In order to do so, the comparison between the losses on the validation and training set was computed, and is reported in Table 1. This was done only to verify that the models were saved w.r.t. the conditions of best train loss and best validation loss, since the train one was computed considering the dropout of the network enabled.

The results cannot be considered very meaningful due to the very unusual losses, but it is quite interesting to notice that in the case of the model with the best validation loss (Best VL), the model found is better also in terms on training error removing the dropout. Moreover, regarding the validation loss, it looks like the model presents a big overfitting, but we need to remember that the final scope is to reduce the loss of the original data and not the one of the augmented version. This is the reason why the final accuracy was not computed with this datasets.

TABLE 1: Losses after training with noisy data

	Best train loss	**Best validation loss**
Training set	13177.574	12747.697
Validation set	41867.780	41470.242

One more interesting analysis was done considering the losses and the accuracy on training and test sets of only the original point clouds. Looking at Table 2, it is interesting to notice that the model with the best training loss provides similar results w.r.t. the model with the best validation loss, leading to a reduced overfitting (test loss is smaller w.r.t. the validation loss in the case with noisy data). About the train loss, the results are not comparable with the previous one since, in this case, the validation set was included in the training one.
Given the point cloud PC_i and the respective output of the network (O_i), the predicted label ($\hat{l}_i$) is taken as:

$$\hat{l}_i = \underset{j}{arg\,max}\, O_i,\ O_i \in \mathbb{R}^{10},\ j \in [0, 9] \tag{1}$$

In this final analysis the training accuracies are high (also with the validation samples inside) and the test accuracies are over 80%, which can be considered acceptable.
One final consideration must be given to the training time: the model is quite simple w.r.t. a lot of cases that can be found in literature, as it allows to reach a good value for the accuracy in a training time of about one hour.

TABLE 2: Comparison between the two models, considering the losses and accuracies without the noisy data of training and test sets.

		Best TL	**Best VL**
Training	**Loss**	18755.615	18696.443
	Accuracy	94.838%	94.688%
Test	**Loss**	34897.816	34794.234
	Accuracy	82.158%	81.938%

After all the previous considerations, it is easy to state that the best model obtained is the one with the best training error, and in Figures 10 and 11 are reported the confusion matrices of the datasets and a graphical example of the outputs.

B. PointNet Model

Before proceeding with the evaluation of the PointNet results, a couple of points must be made. First, considering the original development of the PointNet network, this project used twice the number of points per point cloud. Ideally, this should lead to a better result; however, as can be noticed in literature [9], the PointNet model obtains better results with a smaller dataset (like the 1024 point per cloud used in the paper) or a way bigger number of points (like 4 times the original amount). So, according to these results, it is expected a lower accuracy w.r.t. the data reported in the paper. In addition to this, the noise added originally was gaussian but with different parameters (0 mean and 0.02 as standard deviation), while for the comparison the same dataset of the developed model was used.

TABLE 3: PointNet losses after training with noisy data

	Best train loss	**Best validation loss**
Training set	34.69	1.35
Validation set	118.35	1.34

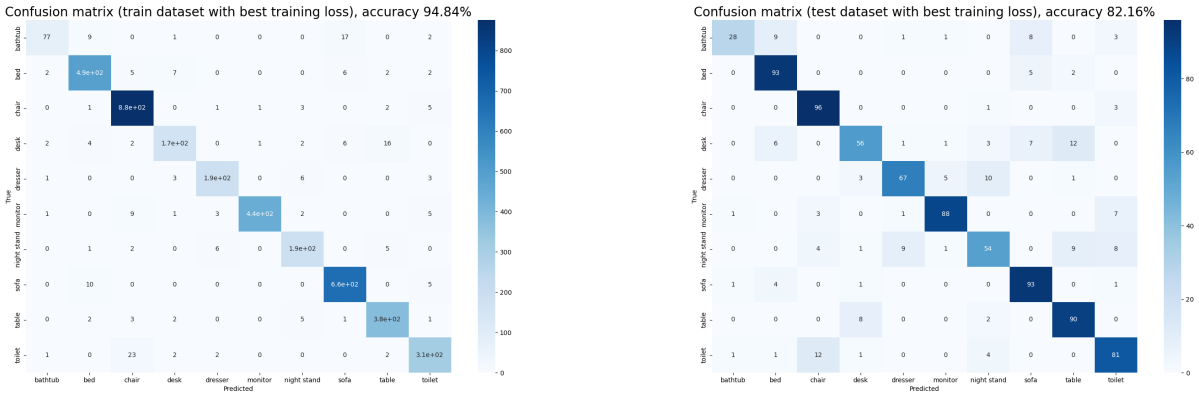
Table 3 reports the best losses obtained during the training sessions of the PointNet model, both for training and validation. As it can be seen, in both cases the values are a lot smaller than the ones obtained by the developed model, but this is due to the difference in the loss and in the encoding for the label using during the training. Regarding the accuracies of these models, presented in Table 4 (referring to PointNet best validation and best training), it can be noticed that the values obtained are lower than the ones presented in the original paper, as expected. Anyway, an interesting aspect is that the results are lower than the accuracy of the developed model. To be noticed that:

- The developed network is simpler than the PointNet in terms of structure, so it's more efficient;
- The results for the PointNet were obtained after with the same number of epochs, datasets and optimizer.

TABLE 4: Comparison between the two models of the PointNet, considering the losses and accuracies without the noisy data of training and test sets.

		Best TL	**Best VL**
Training	**Loss**	44.85	1.23
	Accuracy	79.48%	63.21%
Test	**Loss**	0.75	1.53
	Accuracy	74.44%	49.66%

Last thing to notice is the distribution of the error looking at the confusion matrices. Notice that, even if the false matchings obtained by the models (developed and Pointnet) are somehow expected since they are between similar classes (e.g. *desk*



(a) Train set

(b) Test set

Fig. 10: Confusion matrices of the best model.

classified as *table*), the false matches of the developed model are better distributed across all classes, while the pointnet model, in pariticular for the train dataset, concentrates a lot of the false matches in the class *chair*. Moreover Pointnet completelly fails the classification of the class *night stand*.

VIII. Concluding Remarks

In conclusion, this paper has explained an alternative way of classification for 3D objects from traditionals CNNs or Residual Networks, using a graph representation of the point clouds (then also classifiable as a Graph Classification problem). Taking the single points, we added the estimated normals as node features, together with their positions, and the edges to their 50 closest neighbours. In this way we could use the GCN for convolution, together with some minor MLPs. To be noted how the weight on the one-hot encoding has played a substantial role in the model, together with the dropout in the avoidance of overfitting. Overall this model has the potential to be implemented in more complex forms, such as a Residual Network for Graph Classification, or again 3D Object Classification for heavier datasets.

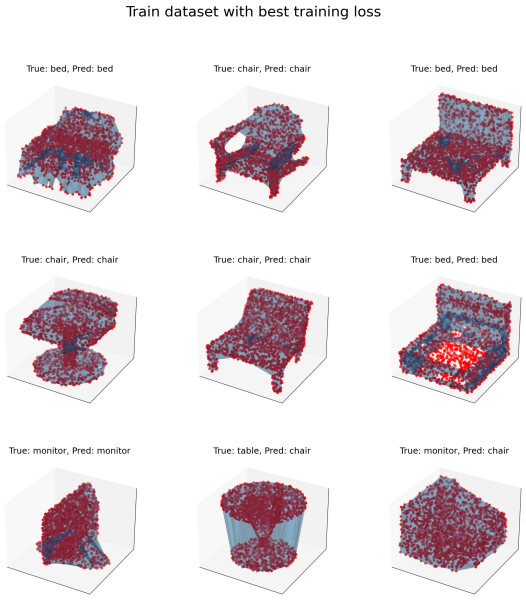
After that, in order to compare the results with an existing model, a PointNet model training was performed with the same dataset. The results obtained confirmed that the developed model is an optimal approach for the Model Classification task, and with further improvement can be compared with other literature's models.

As a conclusion for the project, we acquired more knowledge about the PyTorch environment and Neural Networks techniques. Major difficulties have been encountered in the use of GCN, since they were not in the program of the course, and in the graph construction, since many configurations were tested to better suit the model training.

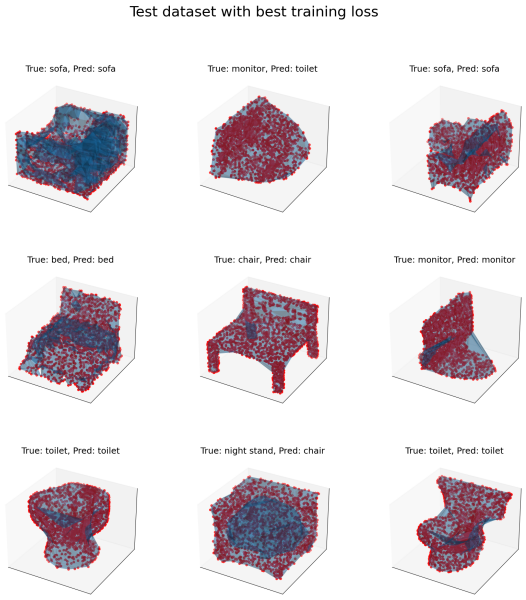
IX. Members contribution

For the first exam date: The workflow was shared within each members that suggested and implemented together all parts of the project.

For the second exam date: Nicola Rizzetto and Gianpietro Nicoletti improved the project adding a better theoretical explanation about the GNN/GCN and implementing a better way to compare the two models (same dataset with same noise, same parameters during the training). Also a comparison of the developed model with different loss and the normalization of the data was added to better explain the design choices.

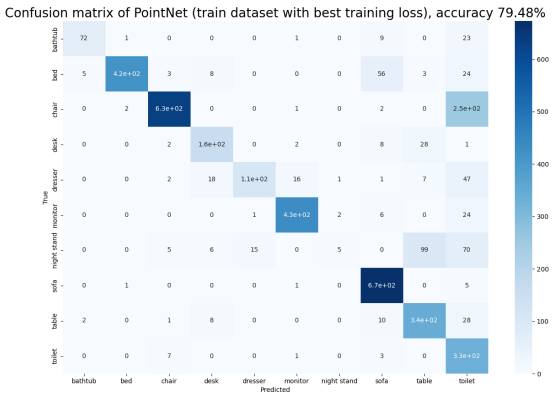


(a) Train set

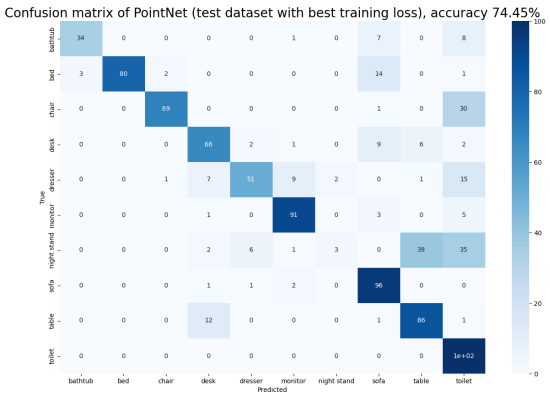


(b) Test set

Fig. 11: Example of the prediction output.



(a) PointNet train set



(b) PointNet test set

Fig. 12: Confusion matrices of PointNet best model.

REFERENCES

[1] N. Sedaghat, M. Zolfaghari, E. Amiri, and T. Brox, "Orientation-boosted voxel nets for 3d object recognition," 2016.

[2] E. Camuffo, D. Mari, and S. Milani, "Recent advancements in learning algorithms for point clouds: An updated overview," *Sensors*, vol. 22, no. 4, 2022.

[3] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *CoRR*, vol. abs/1612.00593, 2016.

[4] S. A. Bello, S. Yu, C. Wang, J. M. Adam, and J. Li, "Deep learning on 3d point clouds," *Remote Sensing*, vol. 12, no. 11, p. 1729, 2020.

[5] F. Vandin, "Learning from networks course 2022/2023,"

[6] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016.

[7] L. Hui and M. Belkin, "Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks," 2020.

[8] T. Cai, S. Luo, K. Xu, D. He, T. Liu, and L. Wang, "Graphnorm: A principled approach to accelerating graph neural network training," *CoRR*, vol. abs/2009.03294, 2020.

[9] D. G. Shayan Hoshyari, Zicong Fan, "Point cloud classification with pointnet,"