UCLouvain

epl

École polytechnique de Louvain

A Comparative Study of 3D Point Cloud Classification: Traditional Method vs. Deep Learning Approach

Author: **Baptiste Rochet**
Supervisors: **Raphaël Jungers, John Lee**
Reader: **Edouard Couplet**
Academic year 2024–2025
Master [120] in Data Sciences Engineering

Acknowledgements

My deepest thanks go to my supervisors, Prof. John Lee and Raphaël Jungers for helping, advising and guiding me in the best possible direction to finalize my master thesis. Thanks also to Prof. Sébastien Jodogne for his support and advice over the past two years.

I would also like to express my gratitude to all the people (especially my family) who supported me throughout my studies at EPL and during the whole project.

Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11 and by the Walloon Region.

Abstract

This master thesis investigates 3D point cloud classification by analysing the differences between traditional methods and deep learning approaches on the ModelNet10 Dataset. Before deep learning became the reference of point cloud classification and segmentation with the emergence of PointNet in 2017 that managed to consume the set of points directly in the network, traditional methods with feature descriptors were used to perform the same task. This thesis focuses on a comparison in terms of accuracy, performance and interpretation between 2 methods for a point cloud classification task: a traditional method based on point feature histogram and PointNet, the reference deep learning method. It also describes the theoretical architecture of these methods and a particular feature selection approach using a margin-based iterative algorithm (SIMBA). Finally, it also presents an in-depth analysis of PointNet++, an enhanced version of PointNet, and a comparison of Farthest Point Sampling, the default sampling algorithm used in PointNet++, and K-Means, an alternative used in its place.

Contents

Introduction

Over the past few years, point cloud technologies have emerged and have become essential in many sectors, from object recognition for autonomous vehicles to virtual and augmented reality. With the rapid development of these technologies, 3D sensors such as lidars and RGB-D cameras have become increasingly accessible and widely commercialized. While 2D image processing technologies have reached maturity, point clouds offer a detailed 3D representation of the world around us.

For a long time, due to the unordered and irregular nature of the points, 3D point cloud data has been processed indirectly by converting it into various forms, such as meshes, multi-view representations or voxel grids. However, in 2017, PointNet [1] revolutionized point cloud processing for classification and segmentation by enabling the direct use of a set of points within a deep neural network. Since the introduction of PointNet, many methods have been developed based on this innovative approach.

The PointNet method belongs to the large family of deep learning methods. And these methods are often regarded as «black boxes». Indeed, it is difficult to interpret how these complicated networks work. Yet all the methods used today in point cloud processing are based on deep learning. So I thought it would be interesting to compare this method with the traditional techniques used previously, both in terms of performance and accuracy and for the interpretation of features on the point cloud.

Prior to PointNet, manually designed rules were used to extract meaningful features, based on statistical and/or geometric properties, such as Conditional random fields (CRF) [2], Markov random fields (MRF) [3] or local descriptors as Scale Invariant Feature Transform descriptor (SIFT) [4] or descriptors defined by feature histograms, such as Fast Point Feature Histograms (FPFH) [5] or Signatures of Histograms of OrientaTions (SHOT) [6].

Given the great number of methods, I have chosen to focus on the technique described in the article «Real-Time Object Classification in 3D Point Clouds Using Point Feature Histograms» [7] which, as its name suggests, uses point feature histograms.

The aim of my master thesis is thus to analyze this traditional method in details and compare it with the deeplearning PointNet approach.

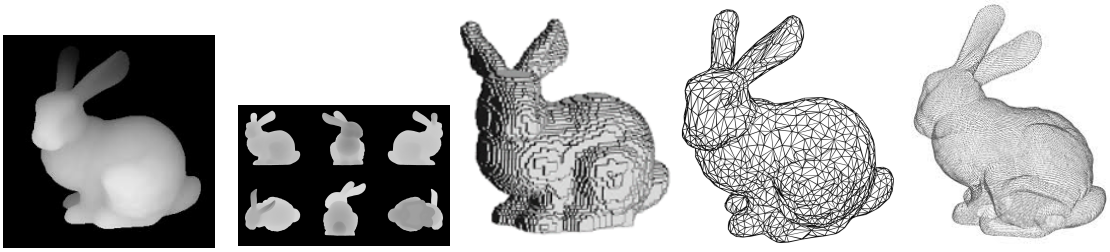
To go further in the comparison I decided, in a second and last stage to have a closer look at PointNet++, the improved version of PointNet, and carry out an in-depth analysis of the various algorithms used in this method. In particular, I chose to explore the impact of K-means algorithm, a variant of the sampling and grouping techniques used by the initial method, to better understand their respective contributions to the processing and classification of 3D point clouds.

Chapter 1

3D datasets and task description

1.1 Point Cloud Data representation

In the field of point cloud processing, various approaches have been developed for representing 3D objects. Here are the most commonly used techniques:



Depth Multi-view Voxels Mesh Pointcloud

Figure 1.1: Data representation

Pixel representation (depthmap)

Pixel representation is the most familiar representation, representing a scene as a series of pixels. To capture information from a point cloud, a 3D scene or object can be projected onto a 2D one, such as a depthmap.

Multi-view representation

Multi-view representation consists in taking several 2D views from different angles, in order to represent a scene in 3D.

Volumetric representation

Volumetric representation is a grid of binary voxels (1 if the voxel is occupied and 0 otherwise)

Mesh

A mesh is a representation composed of vertices, edges and faces that define the geometry of a three-dimensional object.

Point cloud

The point cloud is made up of a multitude of points with 3D coordinates (x,y,z), and sometimes a 4^{th} coordinate such as intensity or color.

Graph

Points can also be represented as the nodes of a graph, taking into account the proximity of points and taking advantage of the geometric structure of the point cloud.

1.2 Applications and tasks description

Point cloud processing has applications in many different fields, reflecting its growing importance in modern technology. In the field of autonomous cars, it can be used to recognize obstacles on the road for safe driving. It can also be used in the manufacturing industry for quality control and industrial process automation. It also aids 3D reconstruction in medical imaging for more advanced diagnostics. It is also used in archaeology, environmental management and many other applications.

Behind all these applications lie a number of specific tasks described below.

Shape classification

Shape classification consists in assigning a category or a predefined label to an object, based on its features.

Object detection

Object detection consists in locating specific objects in an image or a video.

Classification Detection

Figure 1.2: Classification and detection

Segmentation

There are two types of segmentation:

- Semantic segmentation consists in associating each point of the scene with a class label.
- Instance segmentation does the same job as semantic segmentation but also differentiates between instances of the same class.

Semantic Instance

Figure 1.3: Segmentation

Point clouds are also processed for other tasks such as tracking, image recognition, image generation and room layout estimation.

1.3 3D datasets

Dataset	Year	#Samples	Classes	Type	Representation	Label
ModelNet10	2015	4899 objects	10	Synthetic object	Mesh	Object category label
ModelNet40	2015	12,311 objects	40	Synthetic object	Mesh	Object category label
Shapenet	2015	51,190 objects	55	Synthetic object	Mesh	Object /part category label
ScanObjectNN	2019	2,902 objects	15	Real-world object	Points	Object category label
SUN RGB-D	2015	5K frames	37	Indoor scene	RGB-D	Bounding box
S3 DIS	2016	272 scans	13	Indoor scene	RGB-D	Point category label
ScanNet	2017	1,513 scans	20	Indoor scene	RGB-D & mesh	Point category label & Bounding box
KITTI	2016	15K frames	8	Outdoor driving	RGB & LiDar	Bounding box
nuScenes	2020	40K	32	Outdoor driving	RGB & LiDar	Point category label & Bounding box
Waymo	2020	200K	23	Outdoor driving	RGB & LiDar	Point category label & Bounding box
STF	2020	13,5K	4	Outdoor driving	RGB & LiDar & Radar	Bounding box
ONCE	2021	1M scenes	5	Outdoor driving	RGB & LiDar	Bounding box
Semantic3D	2017	15 dense scenes	8	Outdoor TLS	Points	Point category label
SemanticKITTI	2019	43,552 scans	28	Outdoor driving	LiDar	Point category label
SensatUrban	2020	1,2 km^2	31	UAV Photogrammetry	Points	Point category label
SynLiDar	2022	198,396 scans	32	Outdoor driving	Synthetic LiDAR	Point category label
SemanticSTF	2023	2,086 scans	21	Outdoor driving	RGB & LiDar	Point category label

Figure 1.4: Datasets description

Among all existing 3D datasets (some of which are presented in the above Figure), I decided to carry out my research with ModelNet10 [8].

ModelNet10 is a 3D dataset that is widely used in machine learning. This dataset is designed to provide researchers with a comprehensive clean collection of 3D CAD models. The dataset contains synthetic 3D objects and is part of a big dataset (the ModelNet40 dataset). ModelNet10 contains 10 categories of 3D objects:

	Category of object	Train	Test	Total
0	bathtub	106	50	156
1	bed	515	100	615
2	chair	889	100	989
3	desk	200	86	286
4	dresser	200	86	286
5	monitor	465	100	565
6	nightstand	200	86	286
7	sofa	680	100	780
8	table	392	100	492
9	toilet	344	100	444

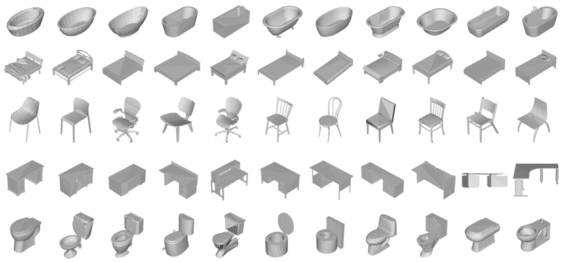


Figure 1.5: ModelNet10 data description

I have chosen this dataset for several reasons:
First of all, because a great deal of classification research has already been carried out and the results obtained will be easily comparable with other methods that have used this dataset.
Secondly, compared to many datasets (especially datasets with real-world objects or scenes), it contains a large number of samples, which contributes to more efficient training and testing.
Thirdly, it is also easy to transform these mesh objects into point clouds: the points can be randomly sampled from the surface of the mesh and the size of the number of points sampled can be adapted. Finally, the ModelNet10 dataset is also perfectly suited for label classification, unlike other datasets which focus on other specific tasks such as segmentation.

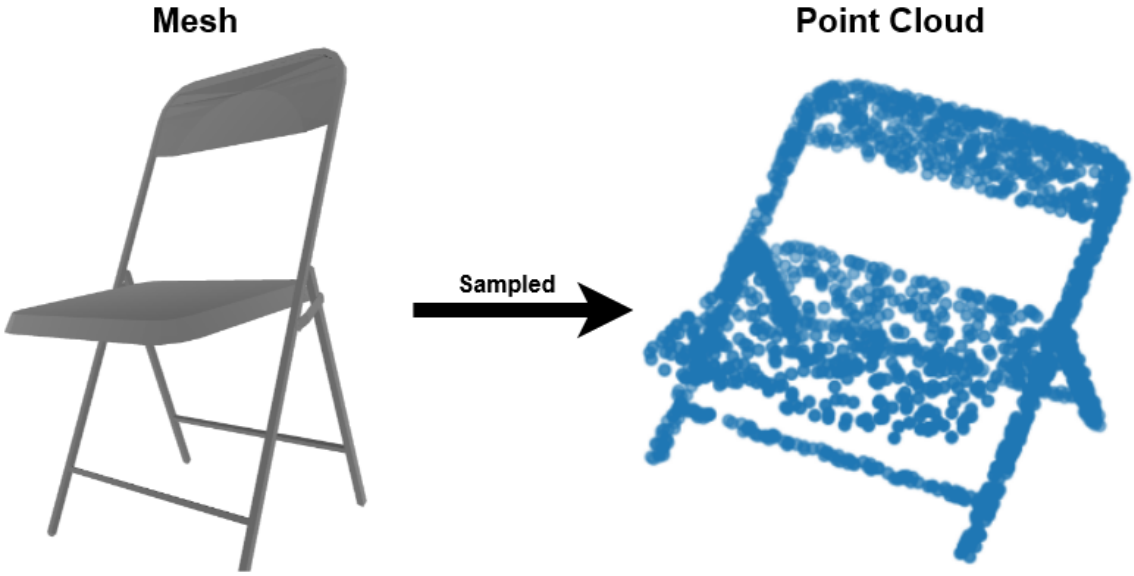


Figure 1.6: Point cloud sampling of 3D mesh of ModelNet10

Dataset creation

For my research, I decided to build a multitude of different datasets to gradually complicate classification. I varied two parameters:

- The number of classes from 2 to 10
- The number of sampled points (respectively 128, 256, 512, 1024 and 2048 points)

The dataset is also normalized between 0 and 1.

1.4 Classification methods

There is a wide range of techniques used to perform point cloud classification or segmentation, each tailored to different data types and applications. The majority of these techniques rely heavily on deep learning methods, which have been proven highly effective for processing and analyzing complex three-dimensional data. These methods exploit deep neural networks to automatically extract features from point clouds and use them to distinguish or segment different objects or regions.

To provide an overview of current approaches, here is a summary table of some of the most commonly used methods in the field:

Méthode	Year	Méthode	Year
3D ShapeNets	2015	GeomCNN	2021
u-net	2015	SimpleView	2021
Voxnets	2015	Pointview GCN	2021
Orientation-boosted voxel nets (ORION)	2016	Polynet	2021
Pointnet	2017	AGCN	2021
Pointnet ++	2017	3D-GAN (apprentissage supervisé)	2021
KD nets	2017	PointNeXt (revisiting Pointnet++)	2022
ECC	2017	Diffusion Unit	2022
O-CNN	2017	point2vec	2023
Pointnet ++	2017	ULIP + PointMLP	2023
Pointnet	2017	ULIP + Pointnet ++ (ssg)	2023
Meshnet	2018	ULIP + PointBERT	2023
G3DNet-18 SVM, Fine-Tuned, Vote	2018	ULIP + PointNeXt	2023
3D SSD	2018	OURS	2023
VoxelNet	2018	SPoTr	2023
Dynamic Graph CNN	2018	Recon	2023
GBNet	2019	OmniVec	2023
InterpCNN	2019	DeLA	2023
3D Points CapsNet	2019	PointConT	2023
Pointflow	2019	PointRAE	2023
VAE-GAN	2020		

Figure 1.7: Classification methods

However, while exploring this field, I also chose to look at traditional methods for classifying point clouds, i.e. those that do not use deep learning. These more classical approaches are often based on geometric, statistical or algorithmic concepts, and can offer advantages in terms of simplicity, interpretability and performance on specific datasets or in contexts where computational resources are limited.

Chapter 2

Traditional methods

The idea of traditional methods is to define a set of local or global features as final feature vector and then give this vector to an SVM [9], Adaboost [10], random forest [11] or K-nearest neighbors [12] for classification. The construction of these global and local features is based on statistical and/or geometric characteristics: there are many methods that extract feature local to a single point as for example 3D Shape Contexts [13] or spin images [14] but the main problem of these methods (that are well suited for object recognition) is that they easily have an increased dimensionality. In fact, we can have 200 to 1000 features for one single point. And that great dimensionality inevitably implies a huge computation time. With this in mind, it was interesting to analyze a method that aims at real-time performance by opting for simpler features. This method is described by M. Himmelsbach et al. in the paper «Real-time Object Classification in 3D Point Clouds Using Point Feature Histograms» [7] and will be detailed in this chapter. This paper outlines a LIDAR-based perception system for ground robot mobility, which includes 3D object detection, classification, and tracking. However, our focus will be solely on the classification aspect. It mainly uses local features extracted from 2 other papers (Lalonde features [15] and Anguelov features [3])which will also be explained in this chapter.

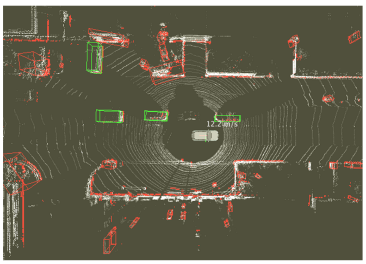


Figure 2.1: Occupancy grid with objects detected by segmentation, represented by 3D bounding boxes [7]

2.1 3D Point cloud classification

Let us start with looking at the process of classifying a 3D point cloud of n points, each defined by its coordinates (x,y,z).

The first step in this process is to analyze the local structure of each point in the cloud by identifying its immediate neighborhood. This step is essential for extracting relevant local features, a procedure commonly referred to as feature extraction.

Once the local features have been extracted, the feature selection phase takes place, which consists in identifying which of the extracted features are the most relevant for effectively differentiating between two sets of points. This step is crucial for improving the performance of the classification model by focusing on the most significant information.

Once the features have been selected, they are used by a classifier, whose task is to analyze these features to determine the appropriate class for each set of points (see Figure below)

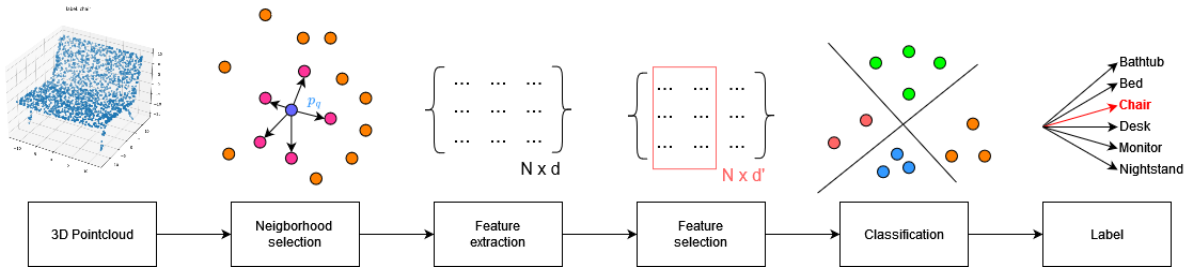


Figure 2.2: 3D Pointcloud classification

Evaluation criteria

The evaluation methods usually chosen for the classification of 3D objects are overall accuracy and mean class accuracy.

The overall accuracy formula for a class i is:

$$Acc_i = \frac{\mathrm{TP}_i + \mathrm{TN}_i}{\mathrm{TP}_i + \mathrm{TN}_i + \mathrm{FP}_i + \mathrm{FN}_i} \tag{2.1}$$

with

- TP_i (True Positives for class i): number of correctly classified instances of the class i.
- FP_i (False Positives for class i): number of times that the model classified an instance as class i while it belongs to another class.

- TN_i (True Negatives for class i): number of times that the model correctly classified an instance of another class as not belonging to class i.
- FN_i (False Negatives for class i): number of times that the model classified an instance as another class while it belongs to class i.

The mean class accuracy for N classes of objects is:

$$mAcc = \frac{1}{N} \sum_{i=1}^{N} Acc_i \tag{2.2}$$

2.2 Feature descriptors

In our case, the point cloud draws all its information from the 3D coordinates of the points (x,y,z). This represents an extremely large amount of data (number of points x3 for each instance). If we have 256 points, this represents 768 different features. The aim here is to find global and local features that best represent the point cloud, in order to give them to a SVM that will perform the classification.

Feature descriptors can be used to represent geometric properties of the neighborhood of a point. There are two 3D shape classification steps :

1. Extract features from an object point cloud capturing the distribution of local and global features.
2. Train a SVM classifier to discriminate class of interest.

2.3 Feature extraction

2.3.1 Global features

Global features are features that do not involve any local computations. In Himmelsbach et al. [7], they called it «object level features».
Let M be the number of points and $i = 1...M$, the 4 object level features are defined as:

- Maximum Object Intensity: $I_{max} = \max I_i$
- Mean Object Intensity: $\mu_I = \frac{\sum I_i}{M}$
- Object Intensity Variance : $\sigma_I = \sum_i (I_i - \mu_I)^2$
- Object Volume V

Since ModelNet10 does not provide information on the intensity or color of points, the various global features associated with intensity will not be used.

2.3.2 Local features

Local features are features that will represent the distribution of the neighborhood of all points in the point cloud. Initially, features will be calculated for all points on the object, but we want to give a representation specific to the object itself, rather than staying within the point level feature. This is why a histogram will be built for each feature, based on the calculation of the feature on all individual points. The feature vector we use will be the concatenation of these local feature histograms and the scalar values of the global features. The histogram is constructed by uniform binning.

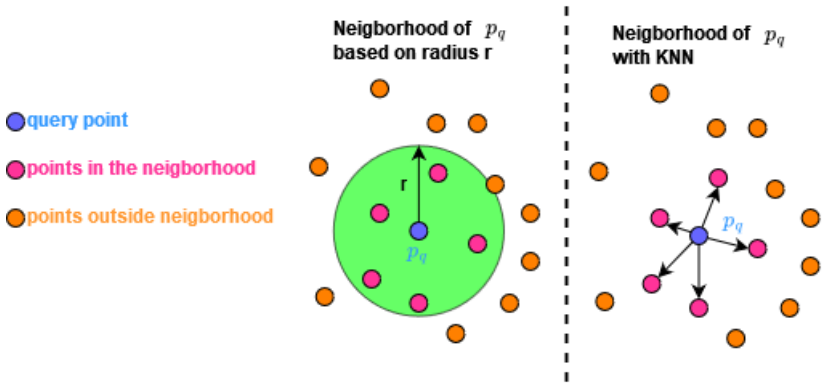


Figure 2.3: Point local features with neighborhood

Lalonde features

The local point statistics are estimated by using the distribution of the 3D points. A decomposition into principal components of the covariance matrix of each point is used to represent this local spatial point distribution.

With a set of points $X_i = (x_i, y_i, z_i)^T$ for $i = 1..M$ and $\bar{X} = \frac{1}{N}\sum_{i=1}^{M} X_i$, let us define the symmetric positive definite covariance matrix for a set of M points:

$$\frac{1}{N}\sum_{i=1}^{M}(X_i - \bar{X})((X_i - \bar{X})^T$$

The principal components of the covariance matrix are the sorted eigenvalues $\lambda_0 >= \lambda_1 >= \lambda_2$ with corresponding eigenvectors $\vec{e_0}, \vec{e_1}, \vec{e_2}$.
The 3 features that they define are linear combinations of the eigenvalues extracted via Principal Component Analysis (PCA). Each feature expresses a very specific spatial distribution of the point neighborhood. These are illustrated below.

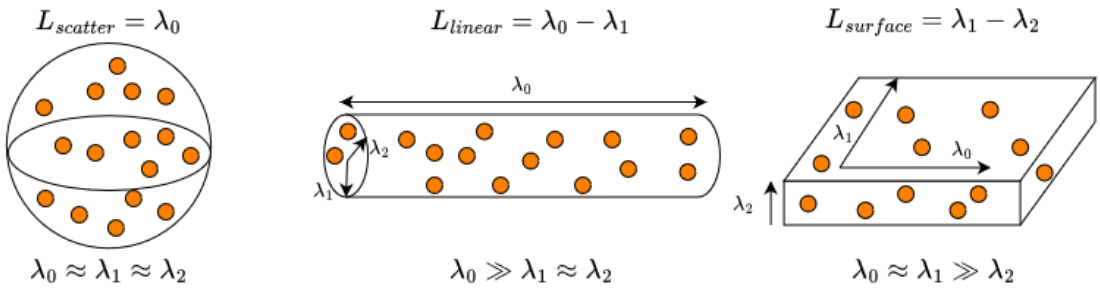


Figure 2.4: Lalonde features

1. When $\lambda_0 \approx \lambda_1 \approx \lambda_2$, nothing can be concluded about the distribution. The points are dispersed or scattered.
   The Lalonde feature of «Scatter-ness»is defined as $L_{Scatter} = \lambda_0$.

2. When $\lambda_0 \gg \lambda_1 \approx \lambda_2$, there is one dominant direction and this reflects a linear distribution of points
   The Lalonde feature of «Linear-ness»is defined as $\vec{L}_{Linear} = (\lambda_0 - \lambda_1)\vec{e_0}$. To calculate the features, we will only take the scalar value of this feature, as we do not need to take the direction into account. The Lalonde feature of «Linear-ness»becomes $L_{Linear} = (\lambda_0 - \lambda_1)$

3. When $\lambda_0 \approx \lambda_1 >> \lambda_2$, there are two dominant directions and this can be likened to a solid surface.
   The Lalonde feature of «Surface-ness»is defined as $\vec{L}_{Surface} = (\lambda_1 - \lambda_2)\vec{e_2}$ and can be reduced to $L_{Surface} = (\lambda_1 - \lambda_2)$.

Let us now look at specific examples of ModelNet10 point clouds to illustrate these Lalonde features.

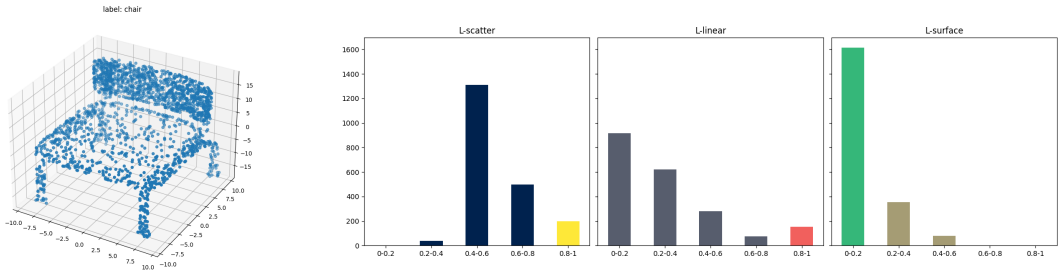


Figure 2.5: on the left: Chair example sampled with 2048 points, on the right: the corresponding histogram of Lalonde

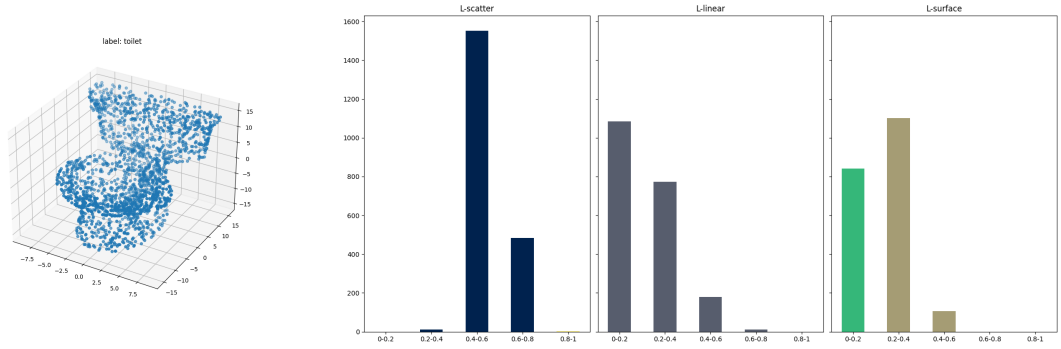


Figure 2.6: on the left : Toilet example sampled with 2048 points, on the right: the corresponding histogram of Lalonde

If we compare the 3 histograms for an example of a chair and toilet respectively, the biggest difference is in the number of points in the [0.8,1] area of the L-scatter feature. We can see that $+/-$ 200 points out of 2048 fall within this zone. This means that λ_0 is very high, and that there is a dominant direction for these 200 points. This is confirmed by the fact that a good proportion of these 200 points are also found in the last bar of the L-linear feature. This means that $\lambda_0 >> \lambda_1$ and thus there is quite a bit of linear structure in the point cloud. Indeed one might think that the points on the chair legs are responsible for this dominant direction in the matrix's eigenvalues. In fact, the toilet, which clearly has no linear structure, has no points in these areas. The last L-surface feature confirms that for the chair, we have $\lambda_0 >> \lambda_1 \approx \lambda_2$

Anguelov features

As described in the paper [7], only one feature of the 2 presented by Anguelov et. al is used. A cylinder of radius r_{ang} and height h_{ang} is constructed around the query point p_q. This cylinder is then divided into n parts A_i of equal size and we count the number of points inside each zone (See on Figure 2.7). We thus add n more histograms to our final feature vector.

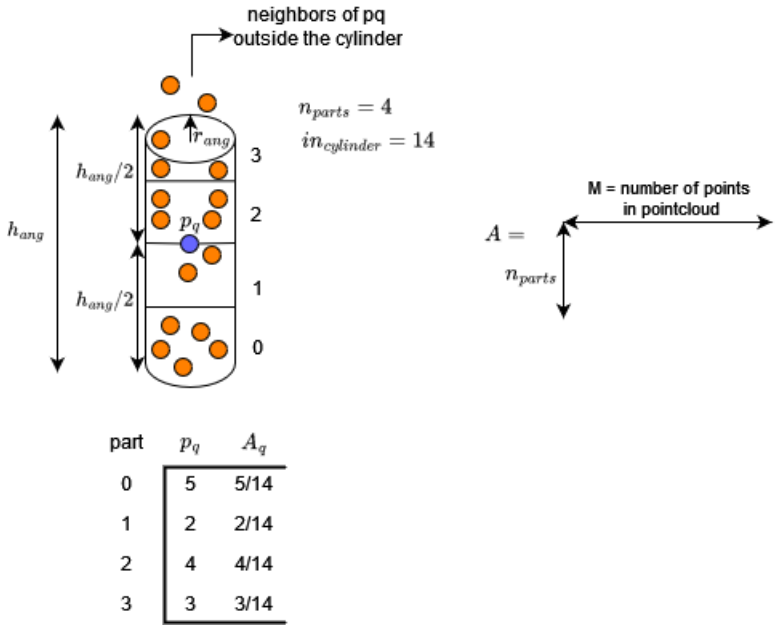


Figure 2.7: Illustration of Anguelov features

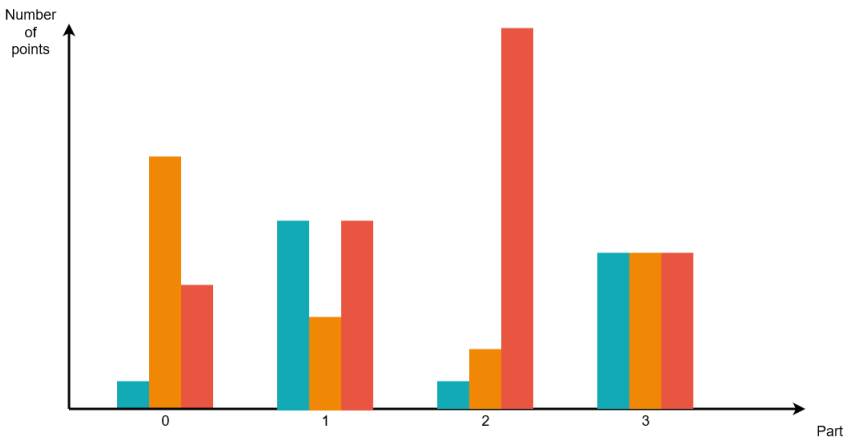


Figure 2.8: Anguelov feature histogram example with 3 bins and 4 parts

2.4 Feature selection based on margin

In the previous sections, various features were identified, but there remains the question of the right number of bins to represent each histogram. We do not want the number of bins to be too high, because then we would have a lot of features which means a higher computation time. And we also do not want the number to be too low, because then the classifier would have too little information to correctly distinguish between the different samples. So we need to strike a balance between accuracy and performance.

It is also important to avoid having many weakly relevant features or redundant features, as this would lead to the curse of dimensionality. In other words, as the number of features increases, the training set will have to grow exponentially in order to generalize as well as possible. This would then become very expensive in terms of computational resources and that is not what we want.

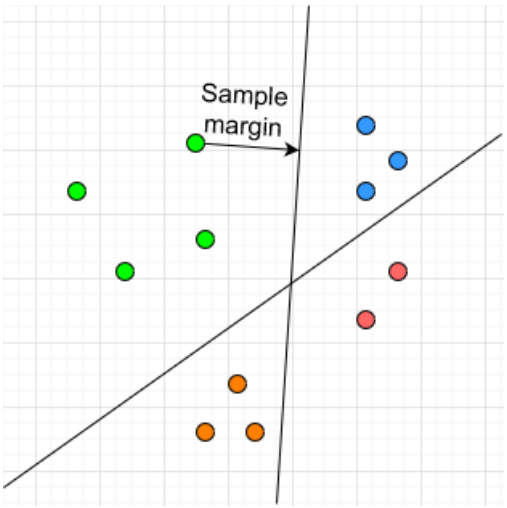
What we want is to choose a subset of relevant features that will enable us to correctly predict the samples in our test set. This is called feature selection.

In this section, we will use an iterative Search Margin Based Algorithm called Simba [16] to estimate how relevant or irrelevant a set of features is.

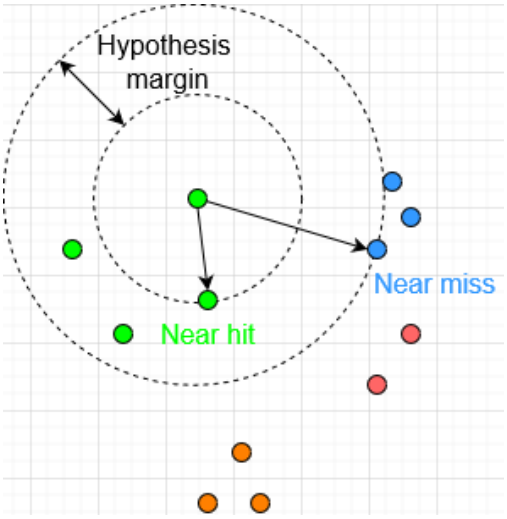
But unlike M. Himmelsbach et al. who already fixed the parameter values of local features and used Simba only to find the number of bins to use in the histogram, we will also use Simba to find the best parameters for the Anguelov and Lalonde features. So we will use Simba to estimate the best radius, the right number of neighbors and the right number of bins for the histogram. Then we will look at the same thing for the Lalonde features (radius, height and number of parts to cut out the optimal cylinder, as well as the number of bins). The point cloud will be normalized between 0 and 1 to also have all parameters between 0 and 1.

In the paper [16], the relevancy of a set of features is determined by the margin it induces. A margin is a geometric measure used to evaluate the confidence of a classifier with respect to its decision. The larger the margin, the more relevant the features are for classifying the dataset. There are several definitions of the margin.

The first one is the *sample margin*, which corresponds to the distance between the classifier's decision boundary and the new instance and the second one is the *hypothesis margin*, it is the maximum distance that a sample point can be moved without changing the label of the new instance (See Figure 2.9).



Sample Margin



Hypothesis Margin

Figure 2.9: Margins

An important fact is that the hypothesis margin lower bounds the sample margin. In other words, the hypothesis margin is the smallest possible margin that can be obtained on the training samples, and any margin observed on these samples will be at least as large as the hypothesis margin. We will be using the hypothesis margin in the SIMBA algorithm. More specifically, the hypothesis margin of an instance f with respect to a point cloud P can be defined as:

$$\theta_p(f) = \frac{1}{2}(\|f - nearmiss(f)\| - \|f - nearhit(f)\|)$$

where:

- nearhit = closest point to f in P with the same label
- nearmiss = closest point to f in P with a different label

Let P be a point cloud, f be an instance and w be the weight vector over the feature set, the margin of f is:

$$\theta_p^w = \frac{1}{2}(\|f - nearmiss(f)\|_w - \|f - nearhit(f)\|_w)$$

where $\|z\|_w = \sqrt{\sum_i w_i^2 z_i^2}$

Normalization requires $\max w_i^2 = 1$

With the training set S, they define the evaluation function as:

$$e(w) = \sum_{f \in S} \theta_{s \setminus f}^w(f)$$

2.4.1 Iterative Search Margin Based Algorithm (Simba)

The goal is to find the weight vector w that maximizes the evaluation function $e(w)$. To achieve this, the algorithm uses a gradient ascent and this can be done because the function is smooth almost everywhere. Let S be the sample of all point clouds instances. The gradient of $e(w)$ evaluated on a feature i is:

$$(\nabla e(w))_i = \frac{\partial e(w)}{\partial w_i} = \sum_{f \in S} \frac{\partial(\theta^w_{S \setminus f}(f))}{\partial w_i} = \frac{1}{2}\left(\frac{(f_i - nearmiss(f)_i)^2}{\|f - nearmiss(f)\|_w} - \frac{(f_i - nearhit(f)_i)^2}{\|f - nearhit(f)\|_w}\right) w_i$$

Algorithm 1 Simba

Let $S \leftarrow$ dataset with features and label of each instance
Let $N \leftarrow$ number of features
Let $M \leftarrow$ number of samples in S
Let $T \leftarrow$ number of iterations of the algo

1. Initialize weights to unit vector $w = (1, 1, ..., 1)$
2. for $t = 1...T$
   (a) Pick randomly an instance f from S
   (b) Calculate $nearmiss(f)$ and $nearhit(f)$ with respect to $S \setminus \{f\}$ and w
   (c) for $i = 1...N$

$$\Delta_i = \frac{w_i}{2}\left(\frac{(f_i - nearmiss(f)_i)^2}{\|f - nearmiss(f)\|_w} - \frac{(f_i - nearhit(f)_i)^2}{\|f - nearhit(f)\|_w}\right)$$

   (d) $w = w + \Delta$
3. $w \leftarrow \frac{w^2}{\|w^2\|_\infty}$ where $(w_i^2) := (w_i)^2$

The time complexity of Simba is $\mathcal{O}(TNM)$.

2.4.2 Point feature histograms

We will determine the optimal parameters of the local features by using Simba.

Bins

As you can see from the graph below, it is not easy to find the optimum number of bins, as it depends heavily on the classes tested, but is also subject to an enormous amount of randomness (due to step 2a) of Simba). So, in a balance between performance and accuracy, I decided to choose a number of bins = 5 for the rest of my research. Indeed, we can see that from 5 bins upwards, the margin is decent, and we can see that increasing the number of bins does not always add a huge margin.

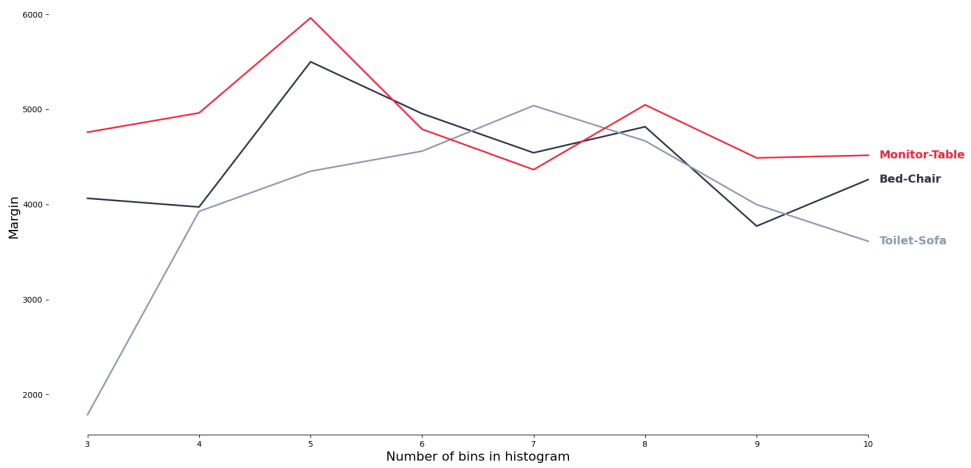


Figure 2.10: Choice of size of the Lalonde histogram

Lalonde parameters

I also used Simba's algorithm to determine the optimal Lalonde radius. In order to do this I calculated the Lalonde feature margins for a set of different datasets with different numbers of points ranging from 2 different classes as performed to determine the number of bins to 10 different classes. All this while varying the number of points between 512, 1024 and 2048 points. It was not possible to take a smaller number of points, as the margin values then obtained become negative, making it more difficult to judge the accuracy of the results.

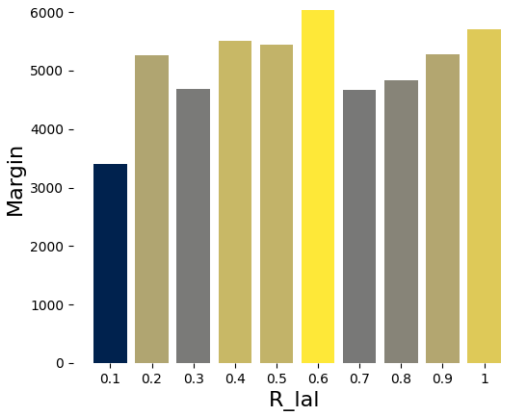


Figure 2.11: Choice of size of the Lalonde histogram

The bars obtained on the figure 2.11 are the means of the margin observed for all datasets. In view of the results of the graph, I fix $r_{ang} = 0.6$

Anguelov parameters

I did the same experiments to determine the optimal values of the Anguelov features. I started by setting the number of bins to 5, as is the case for the Lalonde features. Then I evaluated the following 3 parameters : h_{ang} = the height of the cylinder, r_{ang} = the radius of the cylinder and $npart_{ang}$ = the number of parts dividing the cylinder.

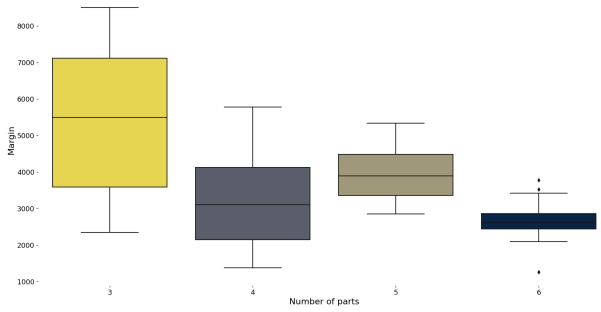


Figure 2.12: Margin analysis with respect to number of parts in Anguelov features for Bathtub-bed data with 512 points

On Figure 2.12, a boxplot of values for 4 different values of n_parts is plotted. Margins are calculated with a number of bins equivalent to 5, a radius ranging from 0.1 to 0.5, and a height ranging from 0.1 to 0.5 on the Bathtub-bed dataset with 512 points. Furthermore, this result was confirmed by analyzing larger datasets. The graph clearly shows that dividing the cylinder into 3 parts gives the best margin. So I fix n_parts_{ang} to 3.

All that remains now is to determine the best parameters for cylinder radius and height (see Figure 2.13). In view of the results of the graph, I fix $h_{ang} = 0.2$ and $r_{ang} = 0.5$

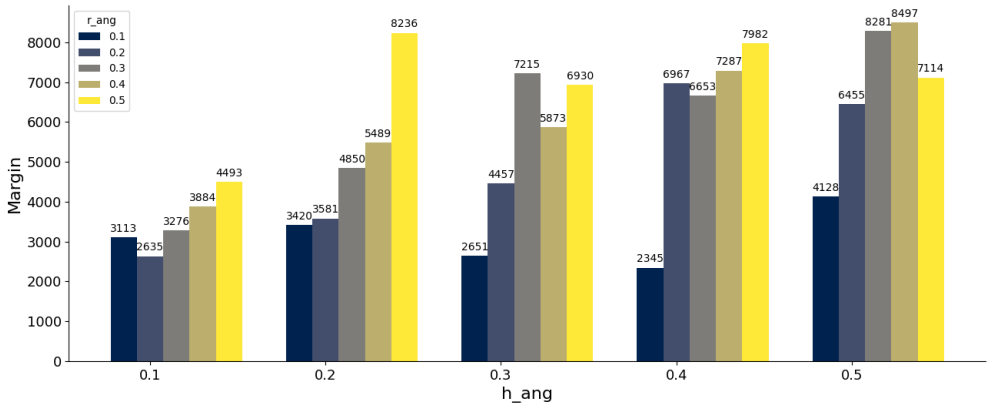


Figure 2.13: Margin analysis with respect to height (h_{ang}) and radius (r_{ang}) of cylinder in Anguelov features for Bathtub-bed data with 512 points

Final feature vector

The final feature vector is then composed of 1 scalar value V from global features, 3 histograms of Lalonde and 3 histograms of Anguelov features each contributing with 5 bins. The final feature vector is :

$$f = (V, H^5_{L_{scatter}}, H^5_{L_{linear}}, H^5_{L_{surface}}, H^5_{A_1}, H^5_{A_2}, H^5_{A_3})$$

The dimension of f is 31.

2.4.3 Simba algorithm accuracy check

After encountering difficulties in determining the best parameters for local features and observing some unexpected results, I decided to check the accuracy of the Simba algorithm. To do this, I used the ModelNet dataset with a variable

number of classes and tested different local feature parameters. I calculated both the margin via Simba and the classification performance of the SVM based on constructed point feature histograms. By analyzing the raw results obtained and constructing several graphs based on these results, I found that the margin was not systematically correlated with the accuracy obtained by SVM.

Here are some typical raw results that I tried to analyse:

Bathtub bed 256 Accuracy

n_part_ang	r_ang	h_ang	2	3	4	5	6	7	8	9	10
3	0,1	0,1	0,76	0,71	0,73	0,76	0,77	0,76	0,77	0,77	0,76
		0,2	0,72	0,67	0,71	0,72	0,72	0,72	0,72	0,71	0,73
		0,3	0,71	0,67	0,71	0,71	0,71	0,71	0,71	0,71	0,72
		0,4	0,70	0,67	0,67	0,71	0,71	0,70	0,71	0,71	0,71
		0,5	0,80	0,67	0,76	0,79	0,79	0,82	0,79	0,79	0,77
	0,2	0,1	0,81	0,68	0,75	0,81	0,77	0,78	0,78	0,81	0,81
		0,2	0,74	0,67	0,79	0,75	0,75	0,77	0,76	0,75	0,75
		0,3	0,80	0,67	0,81	0,81	0,81	0,79	0,77	0,79	0,79
		0,4	0,83	0,67	0,79	0,84	0,83	0,83	0,83	0,83	0,82
		0,5	0,85	0,67	0,80	0,83	0,83	0,86	0,86	0,84	0,86
	0,3	0,1	0,81	0,70	0,81	0,80	0,83	0,83	0,81	0,83	0,83
		0,2	0,81	0,67	0,79	0,80	0,81	0,80	0,80	0,80	0,81
		0,3	0,84	0,67	0,79	0,83	0,83	0,85	0,85	0,84	0,85
		0,4	0,85	0,74	0,83	0,86	0,85	0,86	0,86	0,84	0,88
		0,5	0,85	0,79	0,83	0,87	0,85	0,84	0,84	0,84	0,85
4	0,1	0,1	0,74	0,67	0,73	0,73	0,74	0,74	0,74	0,74	0,74
		0,2	0,71	0,67	0,67	0,71	0,70	0,69	0,70	0,71	0,73
		0,3	0,71	0,67	0,67	0,69	0,69	0,70	0,70	0,71	0,72
		0,4	0,69	0,67	0,67	0,67	0,67	0,67	0,67	0,67	0,70
		0,5	0,71	0,67	0,67	0,67	0,66	0,68	0,69	0,73	0,71
	0,2	0,1	0,80	0,67	0,67	0,75	0,73	0,76	0,75	0,79	0,77
		0,2	0,76	0,67	0,67	0,69	0,69	0,72	0,69	0,74	0,75
		0,3	0,73	0,67	0,67	0,67	0,73	0,74	0,75	0,77	0,75
		0,4	0,81	0,67	0,67	0,72	0,76	0,80	0,81	0,79	0,79
		0,5	0,86	0,67	0,67	0,77	0,83	0,86	0,85	0,84	0,85
	0,3	0,1	0,81	0,67	0,67	0,77	0,72	0,81	0,75	0,79	0,80
		0,2	0,80	0,67	0,67	0,80	0,80	0,85	0,81	0,80	0,81
		0,3	0,82	0,67	0,67	0,70	0,81	0,81	0,83	0,83	0,80
		0,4	0,86	0,67	0,68	0,77	0,84	0,84	0,85	0,84	0,84
		0,5	0,86	0,72	0,80	0,80	0,84	0,85	0,85	0,85	0,85
5	0,1	0,1	0,78	0,67	0,75	0,77	0,79	0,79	0,79	0,79	0,79
		0,2	0,74	0,67	0,73	0,73	0,73	0,73	0,73	0,74	0,75
		0,3	0,75	0,67	0,72	0,73	0,74	0,76	0,75	0,74	0,74
		0,4	0,73	0,67	0,71	0,72	0,73	0,71	0,72	0,73	0,72
		0,5	0,81	0,67	0,67	0,75	0,75	0,79	0,77	0,79	0,78
	0,2	0,1	0,82	0,71	0,76	0,78	0,75	0,77	0,75	0,81	0,81
		0,2	0,79	0,67	0,81	0,79	0,80	0,81	0,79	0,81	0,81
		0,3	0,83	0,67	0,78	0,83	0,84	0,83	0,83	0,83	0,81
		0,4	0,83	0,67	0,74	0,83	0,83	0,87	0,83	0,83	0,85
		0,5	0,85	0,67	0,75	0,79	0,83	0,85	0,87	0,85	0,84
	0,3	0,1	0,81	0,67	0,83	0,83	0,84	0,83	0,80	0,80	0,80
		0,2	0,81	0,67	0,77	0,83	0,83	0,81	0,81	0,81	0,81
		0,3	0,82	0,67	0,78	0,82	0,81	0,83	0,85	0,83	0,83
		0,4	0,83	0,67	0,80	0,85	0,81	0,83	0,83	0,83	0,85
		0,5	0,84	0,67	0,79	0,84	0,86	0,87	0,86	0,85	0,85

Bathtub bed 256 Margin

n_part_ang	r_ang	h_ang	2	3	4	5	6	7	8	9	10
3	0,1	0,1	1354	1100	1107	1133	1278	1289	1355	1405	1288
		0,2	1349	883	910	1533	1429	1178	1322	1216	1296
		0,3	1195	1026	1184	720	1493	1243	1396	1276	1302
		0,4	1309	696	1032	1229	1373	1252	1144	1163	1047
		0,5	1615	1001	1411	1135	1233	1480	1154	1475	1473
	0,2	0,1	1517	1088	1308	1405	1535	1613	1520	1539	1428
		0,2	1702	1612	1300	1446	1665	1928	1122	1629	1416
		0,3	1019	1869	1633	1124	1408	1333	1074	1186	1080
		0,4	903	886	1514	1663	1424	1144	1684	1289	1056
		0,5	2006	1338	2271	2509	2713	2446	2492	2700	1701
	0,3	0,1	1320	1403	1113	1364	1562	1491	1536	1234	1275
		0,2	1189	2034	1662	1167	2146	1801	1795	1483	1388
		0,3	1401	1799	1564	1658	1583	2164	1631	1494	1592
		0,4	1953	1526	2283	1937	2553	1490	1994	1703	2428
		0,5	2660	1962	2076	1957	2932	2771	2879	2435	2358
4	0,1	0,1	819	691	591	906	759	913	1106	1069	992
		0,2	918	558	604	719	847	903	995	743	780
		0,3	800	735	476	589	883	1007	811	872	984
		0,4	850	801	626	772	1092	1145	1073	1103	1424
		0,5	1344	930	703	685	753	911	886	1418	1237
	0,2	0,1	1451	971	613	854	1347	1119	1251	1363	1174
		0,2	1417	1107	587	1017	749	945	1256	1012	1003
		0,3	1142	890	813	971	987	1109	1459	862	983
		0,4	1007	650	806	1061	1079	1331	1145	1033	756
		0,5	1880	1255	720	1303	2096	1750	1835	2032	2187
	0,3	0,1	939	1021	779	822	868	1191	894	922	1004
		0,2	1826	931	1393	1258	1763	1474	1263	1260	1311
		0,3	1338	882	1162	1091	1016	1290	1415	1184	1638
		0,4	1619	1324	769	1033	1444	1484	1982	1955	1587
		0,5	2518	1385	1060	1633	1741	2176	2287	2716	2650
5	0,1	0,1	1286	997	1167	1250	1306	1231	1534	1285	1389
		0,2	1270	860	1041	1144	1105	1542	1214	1681	1424
		0,3	1534	1053	1075	1079	1553	1287	1356	1496	1259
		0,4	1441	1074	1134	1270	1658	1657	1632	2109	2163
		0,5	1509	1135	1188	1457	1298	1252	1616	1455	1760
	0,2	0,1	1989	1187	1517	1496	1661	1571	1629	1596	1957
		0,2	1607	1145	1883	1524	1276	1650	1217	1085	1442
		0,3	1603	911	1216	1493	1254	1270	1791	1888	2153
		0,4	1579	1099	1510	1531	1628	1768	1414	1671	1793
		0,5	2277	1767	1630	1721	1721	1625	2351	1983	1833
	0,3	0,1	1612	1449	1087	1415	1491	1310	1317	1709	1205
		0,2	1794	1416	1845	1410	1796	1484	1630	1846	2689
		0,3	1383	1226	1775	2061	1193	1810	2107	2433	2320
		0,4	2131	1174	2196	2365	1156	2098	2142	1823	2472
		0,5	2488	1713	1983	2198	1847	3005	1962	3004	3532

Figure 2.14: Raw results of bathtub bed dataset with 256 sampled points of accuracy and margin of Anguelov features obtained with SVM and SIMBA algorithm respectively

These data are organized as follows :

- the columns correspond to the number of bins from 2 to 10

- each row is a combination of a 3 Anguelov parameters

    1. n_part_{ang} : number of parts the cylinder is divided
    2. r_{ang} : the radius of the cylinder
    3. h_{ang} : the height of the cylinder

So each value in the table represent the result for a single combination of 4 parameters (it is a pivot table). On the left: SVM accuracy results. On the right: the margins calculated via SIMBA algorithm.
In purple: the maximum values for accuracy and their corresponding values of margin.
In blue: the maximum value of margin.

If we look only at these values, we can see that the maximum margin does not correspond to the maximum accuracy. However if we check for some values, there still seems to be a correlation between margin and accuracy. In order to check the correlation, we will generalize this table by isolating one parameter at a time and calculating the aggregated means according to the number of bins.

Let us have a look at the table generated by isolating Anguelov N_{part} parameter

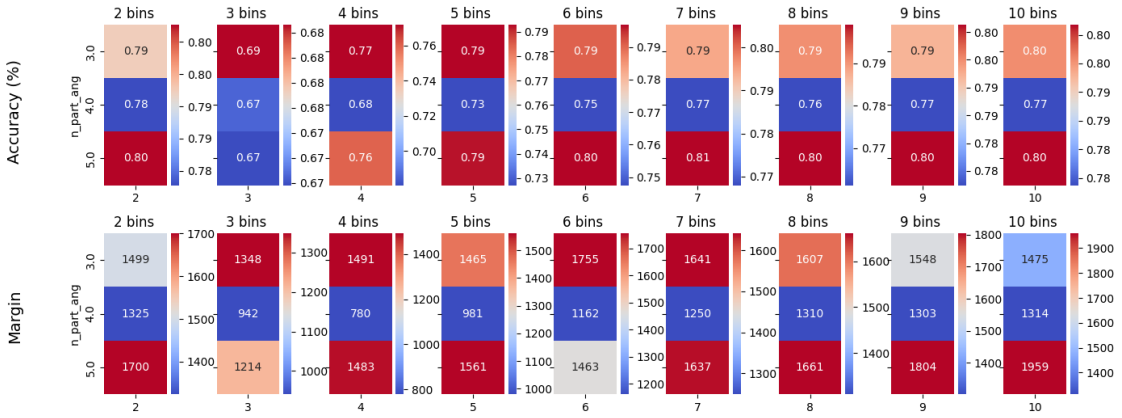


Figure 2.15: Comparison of margin and accuracy wrt to the number of bins for choice of the n_part parameter of Anguelov with the dataset bathtub-bed with 256 sampled points

The above figure gives us a lot of information: it is clear that $n_{part} = 4$ is a really bad choice, both margin and accuracy are much lower for this choice. We also see that there is a great correlation between margin and accuracy as great values in margin are also great in accuracy.

When it comes to determining the optimal value of a parameter, it is crucial not to lose sight of the fact that the search for this value is based on a dual objective: maximizing model performance while minimizing computation time. These two aspects must be balanced to obtain a result that is not only efficient in terms of accuracy, but also viable in terms of computational costs.

The best choice for n_{part} for this dataset would be 3: fewer histograms to calculate and a great performance.We can see that starting from 5 bins we reach an accuracy plateau for $n_{part} = 3$. So it is best to choose a lower number of bins if we can capture the necessary information with them. The logical choice is then 5 bins.

The same selection procedure was used to evaluate the best parameters for the cylinder radius and height. However, if we look at datasets with more or fewer classes and with a different number of sampled points, we observe that it is very difficult to choose a combination of parameters that is suitable for all datasets. It would be necessary to perform the analysis for all datasets one by one and define the best parameters each time, but this would take an enormous amount of time, especially for datasets with a large number of samples.

I also assessed the correlation between accuracy and margin as a function of the number of bins (see Figure 2.16). The Figure shows that the correlation is effective (Pearson correlation coefficient = 0.65) for bathtub-bed dataset. The Figure 2.17 however, shows the correlation comparison for the bathtub-bed-chair-desk dataset with 256 sampled points and here we see that in spite of the clear correlation for 3 and 4 bins, correlations reach negative values for the most bins (Pearson correlation coefficient = 0.29).

These results finally confirmed my initial impression: the relationship between margin and accuracy is highly variable and varies according to the number of classes, the number of points sampled and the parameters used.

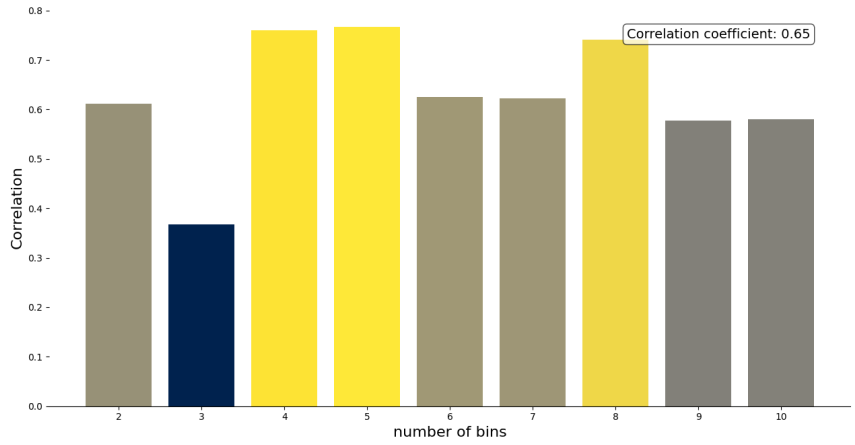


Figure 2.16: Correlation between accuracy and margin as a function of the number of bins for bathtub-bed dataset with 256 sampled points

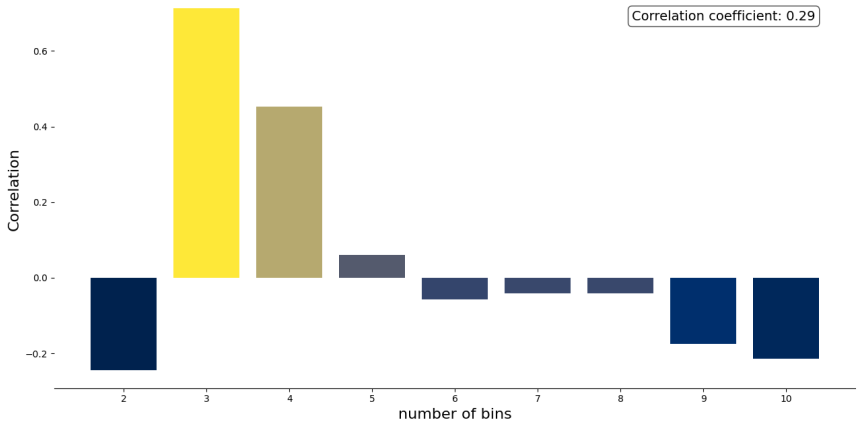


Figure 2.17: Correlation between accuracy and margin as a function of the number of bins for bathtub-bed-chair-desk dataset with 256 sampled points

In view of both correlation between margin and accuracy and the difficulty of choosing the best combination of parameters, it is hard to have complete confidence in the SIMBA algorithm, but it does give us some good pointers on the choice of parameters. That is why I decided to set reasonable values for the parameters, again with an eye to both performance and computation time.

Final choice of local parameters

After these checks, I finally chose the following parameters for the Lalonde features:

- $r_{lal} = 0.5$
- $nbr_{bins} = 5$

and the following parameters for the Anguelov features:

- $N_{part} = 3$
- $h_{ang} = 0.5$
- $r_{ang} = 0.3$
- $nbr_{bins} = 5$

2.5 SVM for classification

Once the final feature vector has been constructed, it now needs to be given as input to a classifier. The classifier selected for this study is the well-known Support Vector Machine (SVM) classifier [9] but we could also have used an Adaboost model or a KNN. To identify the most appropriate parameter values, a grid search was performed. This technique made it possible to vary the penalty parameter C as well as the value of the kernel coefficient γ while using the RBF (Radial Basis Function) kernel. This approach optimized model performance by finding the parameter combination best suited to the data. The best parameters found are $C = 1$ and $\gamma = 0.0001$.

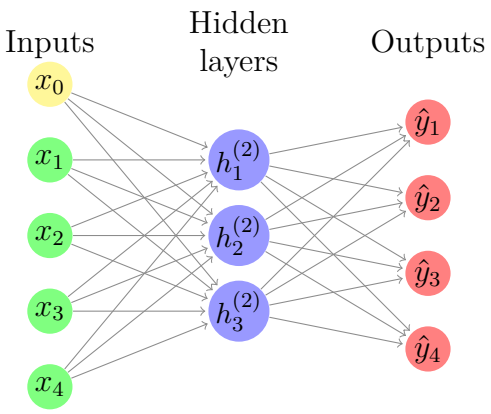
Chapter 3

Deeplearning: PointNet

Before the PointNet method emerged in 2017, the point cloud was systematically transformed into a regular format such as a volumetric representation, a mesh or a multi-view representation. Point clouds are not evenly distributed in space (i.e. it is possible to have a huge density of points in one place and a minority in another) and they are randomly sorted without any link between them. In fact, you could have a specific point on the rabbit's ears and then a point on its tail. This irregularity and disorder meant that pointclouds could not be directly processed by the deeplearning methods of the time. Charles R. et al. then found a way to give an unordered pointcloud as input to a neural network. In the paper [1], they presented both a theoretical demonstration and an experimental analysis of the results obtained.

3.1 Multilayer perceptron (MLP)

PointNet is essentially made up of several chained MLPs, so it is a good idea to start by defining what an MLP is. A multi layer perceptron is a type of artificial neural network composed of fully connected layers of neurons including input layer, hidden layers and an output layer. The hidden layer and the output layer apply transformations followed by a nonlinear activation function that enables to capture complex relationships. This allows the model to go beyond linear relationships.



3.2 Point Cloud properties

The way PointNet works is based on three major features of a set of points: Let P be a set of n points in R_n

1. *Order Invariance*: A point cloud is a set of points with no established order, whereas pixel sets in an image, or voxel grids in a volumetric representation are well-ordered sets allowing easy processing. Indeed, the position of a pixel in an image in the grid corresponds to the pixel's position in the image. This proprety implies that the network needs to be invariant to the permutation of the points (i.e. invariant to n! permutations)

2. *Interaction among points*: the points belong to a distance-metric space, indicating that the points are not isolated. Instead, the point's neighborhood contains its most important information. Indeed, a single point whose only information is its x,y,z coordinates gives us no relevant information to describe the whole set of points. It is the local interaction between points that will provide our neural network with relevant information.

3. *Transformation Invariance*: when processing a point cloud, any transformation such as point rotation or point translation must not change the class in which the point cloud is classified.

3.3 PointNet Architecture

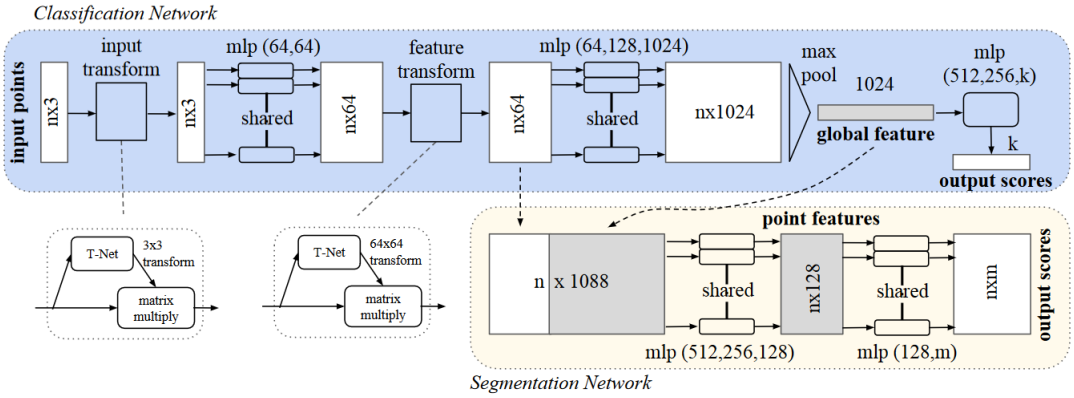


Figure 3.1: PointNet Architecture (Charles R. Qi et. al. [1])

PointNet's architecture can be seen in Figure 3.1, and we will describe the classification part in more details below.

Since the input is unordered, the solution found by the author is to use a *symmetric function* (in this case, the max pooling function) to aggregate the information from all the points. A symmetric function takes a vector of n points as input and returns a vector that is invariant to the input order.

A symmetric function can be defined as a function that always returns the same value no matter the order of the arguments (no matter any permutation π_i). For example $f = max$ or $f = \sum x_i$.

$$f(x_1, x_2, .., x_n) \equiv f(x_{\pi_1}, x_{\pi_2}, , .., x_{\pi_n}) \tag{3.1}$$

The objective is to develop a family of symmetric functions using neural networks. The approach involves approximating a general function defined over a set of points by applying a symmetric function to the transformed elements within the set.

$$f(x_1, x_2, .., x_n) \approx g(h(x_1), h(x_2), .., h(x_n)) \tag{3.2}$$

The author has observed that, in the situation described below, f is symmetric if g is symmetric.

$$f(x_1, x_2, .., x_n) \approx \gamma \circ g(h(x_1), h(x_2), .., h(x_n)) \tag{3.3}$$

This enabled him to build a first version of PointNet: *PointNet vanilla* (See Figure 3.2).

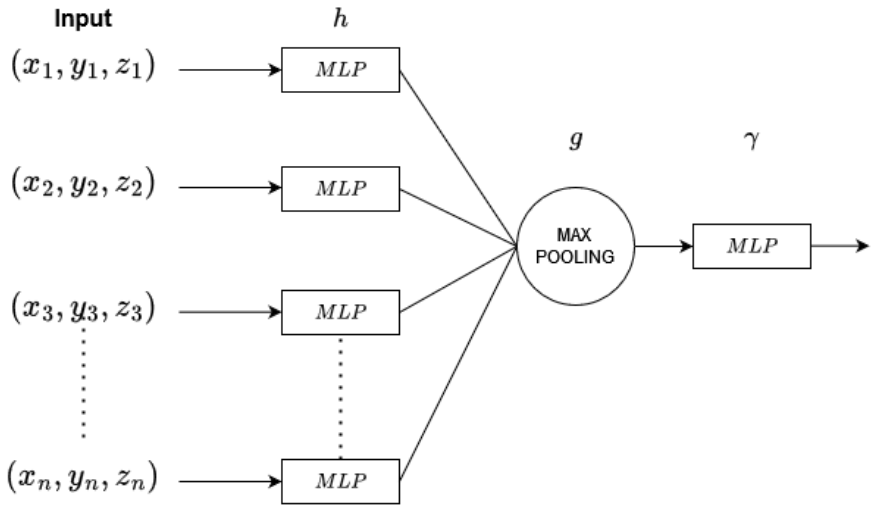


Figure 3.2: PointNet Vanilla

Description of **PointNet Vanilla** architecture:

- The function h projects raw points into a high dimensional embedding space. This is approximated by a multi-layer perceptron (**MLP**).
- The function g (**Max pool**) extracts global features from the point cloud's embedding. The author chooses the max pooling symmetric function. He tested with average pooling and weigthed average pooling but he observed that max pooling achieved the best results. The max pooling function calculates the maximum value of parts of a feature map in order to downsample the feature map (see Figure 3.3).
- The function γ (**MLP**) will apply an additional transformation to optimize the global representation for improved classification accuracy.

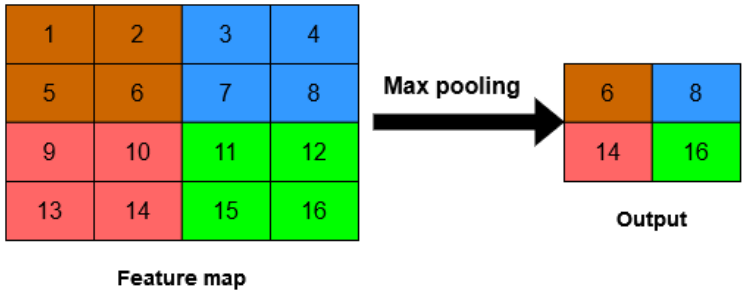


Figure 3.3: 2x2 Max pooling function example

PointNet (vanilla) as defined in Figure 3.2 does not respect the 3rd property (transformation invariance), it is still variant under rigid transformations. The simplest way to do this is to align all input set to a canonical space before feature extraction. In other words, before meaningful features are extracted from the data, the inputs are aligned or normalized to minimize variations due to differences in orientation, position or scale between data sets. This ensures that feature extraction is relevant and not based on arbitrary small variations or orientations in the input data.
The **transformation net layer (T-net)** is used to normalize the input data points using an affine transformation matrix (i.e. a combination of rotations, translations and possibly other linear transformations). This network is a miniature version of the entire PointNet network.

The second property (point interaction) relates to the segmentation part so it will not be discussed.

3.4 Hyper-parameter selection

Ideally, hyper-parameter fine tuning should be used to identify the best parameters. The various hyper-parameters we can play with to improve model performance are defined below:

1. Optimizer = defines how model weights and biases are updated in the training process
2. Number of epochs = the number of algorithm runs during training phase
3. Batch size = the number of samples propagated through the neural network
4. Learning rate = parameter that defines the step size of gradient descent. It therefore influences the evolution of model weights and how fast we will converge to minimum loss and optimal weights.
5. Batch normalization momentum (bn momentum)

I decided to set these hyper-parameters to reasonable values for all datasets after exploring several possible configurations. The results obtained are surely not optimal, but they already provide a solid basis for comparison with those obtained by the traditional method using local descriptors. Here are the values used to calculate all future results. For the choice of optimizer, learning rate, batch size, these are the values specified in article [1] to obtain the results for ModelNet40.

1. Optimizer: Adam optimizer
2. Number of epochs = 100
3. Batch size = 32
4. Fixed learning rate = 0.001
5. Fixed batch normalization momentum = 0.5

Unlike Charles et.al, who use a decay rate for batch normalization that starts with 0.5 and gradually increased to 0.99, and an adaptive learning rate (divided by 2 every 20 epochs), I decided to fix the learning rate and the bn momentum. I also chose a number of epochs of 100, which I thought was reasonable for PointNet to converge for most datasets.

Chapter 4

Deep Learning: PointNet++

PointNet++ [17] is an enhanced version of PointNet [1], widely used in many fields of application related to 3D point clouds. In our case, this method is particularly interesting for its 3D point cloud classification and segmentation capabilities, although only the classification part will be presented here.

PointNet++ has been developed to overcome one of PointNet's main limitations: its inability to capture local structures. This weakness reduces PointNet's ability to identify fine-grained patterns and generalize efficiently to complex scenes. PointNet++ introduces hierarchical processing to capture local structures by subdividing point clouds into successive local regions.

Since the appearance of PointNet and PointNet ++ in 2017, several derivative methods have been developed to improve their performance and address their limitations. These include PointWeb [18] (2019) which offers an Adaptive Feature Adjustment module to improve interaction between points and solve problems of information exchange and feature refinement, DGCNN [19] (2019) which uses graphs to model relationships between points and enrich local features and RandLA-Net [20] (2021) which uses random point sampling and introduces a new local feature aggregation module for preserving geometric features. We can also mention Point-BERT [21] or Point-MAE [22] that are also derived from Pointnet++.

Pointnet ++ can be used for a wide range of applications, such as classification, segmentation, navigation for autonomous robots, or analysis of specific structures (e.g. PAN: Improved PointNet++ for Pavement Crack Information Extraction [23] extracts accurate pavement crack information in order to help with routine maintenance and reduce the risk of traffic accidents).

4.1 Hierarchical Point Set Feature Learning

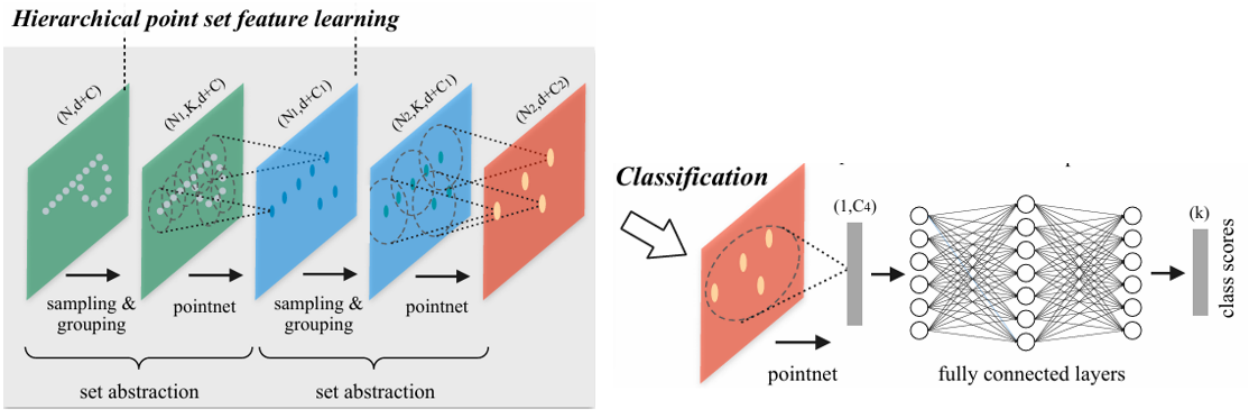


Figure 4.1: Hierarchical Point Set Feature Learning architecture [17]

Charles R. et al. first introduce Hierarchical Point Set Feature Learning. The principle of this hierarchical neural network is to divide a point cloud into local regions (sampling and grouping step), extract local features from these regions (PointNet step), then repeat this process in a hierarchical fashion, progressively abstracting larger and larger local regions.

Here, N represents the total number of points in the point cloud, d denotes the spatial dimension (typically x, y, z coordinates), and C refers to additional features associated with each point, such as intensity or color.

Once we have reduced the abstraction to a small number of points, we use PointNet again to generate a final feature vector of dimension $(1, C')$. This vector is then passed through a fully connected layer, which produces the scores associated with the different classes. These scores then allow us to classify the initial point cloud into the most relevant class.

4.1.1 Sampling layer

The idea is to divide the set of points into local regions of points by selecting specific points that will be the centroids of these regions. To achieve this, the farthest Point Sampling algorithm (FPS) is used. For the same number of centroids, FPS offers more uniform coverage of the point cloud than a simple random sampling.

Iterative Farthest Point Sampling (FPS)

Algorithm explanation (see Figure 4.2 for illustration):

1. At beginning (when no centroids are sampled)
    - Pick randomly a first centroid C_0 from the point cloud.
    - Calculate the distances from all points to the centroid.
    - The farthest point from the first centroid (global maximum of the distances) becomes a new centroid (C_1).
2. When several centroids are sampled:
    - Calculate the distances from all points remaining (all points that are not sampled) from the last chosen centroid.
    - For each point remaining, take the smallest distance separating it from a centroid (local minima).
    - The point with the higher distance (global maximum of local minima) becomes a new centroid.

    Repeat this step until M centroids are sampled.

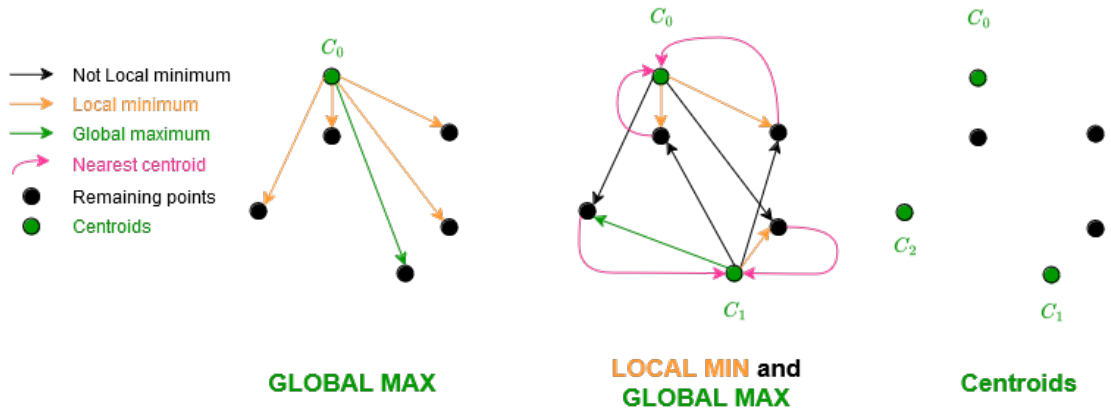


Figure 4.2: Farthest Point Sampling illustration

4.1.2 Grouping layer

Once the points have been sampled, the grouping layer finds the neighborhood of all the centroids to create local regions of points. To achieve this objective, the authors of the paper [17] used two methods, which are described below.

Ball Query

Ball query method allows to identify neighboring points within a specific radius around a centroid by computing the Euclidean distance

$$d(c, p_i) = \sqrt{(x_{p_i} - x_c)^2) + (y_{p_i} - y_c)^2) + (z_{p_i} - z_c)^2)}$$

which measures the distance between the centroid c and a point p_i in the point cloud. Given a chosen radius r, a point p_i belongs to the neighborhood of a centroid c if $d(c, p_i) < r$. In the implementation, the number of neighbors per local region is bounded below K. However, the local number of neighbors may be smaller than K, making K variable and centroid dependent.

K-Nearest-Neighbors (KNN)

KNN is an alternative query for finding neighborhood of centroid where we fix the number of neighbors K and find the K-nearest neighbors. The method for finding neighbors is simple: calculate the Euclidean distances between the centroid and all points and select the K smallest distances.

This method allows to have a fixed number of neighbors, but if the distribution of points is not uniform, it can end up with neighbors that are very far apart.

The ball query method, on the other hand, ensures a fixed local region size regardless of point density which is preferable for local pattern recognition tasks because it enables the extraction of more consistent and transferable local features across space. This is why Charles R. et al. chose to use the ball query method rather than KNN. However, it should be stressed that the impact of this decision may vary according to the data used and the nature of the task.

4.1.3 PointNet layer

At the end of the sampling and grouping stage, we obtain N' local regions of K neighbors (with K varying according to the region) to which we will apply a mini-PointNet.

In this phase, the coordinates of the points in each region are expressed in relation to the local centroid. Thus, for each point p_i with $i = 1..K$ located around the centroid c, the relative 3D coordinates are calculated as follows:

$$x_{p_i} = x_{p_i} - x_c, y_{p_i} = y_{p_i} - y_c, z_{p_i} = z_{p_i} - z_c$$

On the input side, the data are represented by a tensor of dimensions $N' \times K \times (d+C)$. At the output of this layer, each local region is represented only by the features

of its centroid and those describing its neighborhood. The output therefore has a dimension of $N' \times (d + C')$.

4.2 PointNet ++: Non-uniform sampling density

The challenge with Point Set Feature Learning is that it often happens that the point cloud is not uniformly dense i.e. with some parts extremely sparse and others extremely dense. This implies that features learned in dense point cloud may not generalize well for a sparse point cloud.

To address this, Charles R. et al. introduced a robust feature learning with density adaptive PointNet Layers. They called this hierarchical network: PointNet++. In this network, each abstraction layer captures local patterns at multiple scales and integrates them adaptively based on the density of the points in the region There exists two types of density adaptive layers that are illustrated in Figure 4.3.

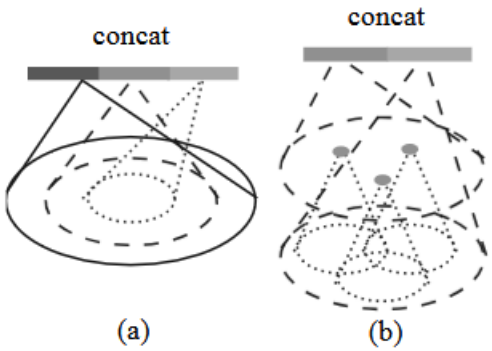


Figure 4.3: (a) Multi-Scale grouping (MSG) (b) Multi-Resolution grouping (MRG) [17]

Multi-Scale grouping (MSG)

This method groups points around a centroid on several scales, using several different radii. The network is trained to develop an optimized approach for combining multi-scale features by randomly dropping out input points with a randomized probability for each instance (*random input dropout*).

- Pros: ability to capture complex local structures
- Cons: computationally expensive, as it runs PointNet for different scales at each centroid.

Multi-Resolution grouping (MRG)

The features of a region at a given level L_i are obtained by concatenating two vectors. The first vector summarizes the features of each sub-region of the lower level L_{i-1}, using the set abstraction level. The second vector is derived by directly processing all raw points in the local region using a single PointNet. MRG is faster than MSG since feature extraction in large-scale neighborhoods at the lower levels is avoided.

SSG vs. MSG

PointNet++ offers two main variants for grouping points: SSG (Single Scale Grouping) and MSG (Multi-Scale Grouping). These approaches differ in the way they deal with the local characteristics of the point cloud. SSG extracts local information using a single scale, while MSG combines local features at multiple scales, better capturing the complexity of areas of non-uniform density. Experimental results on Modelnet10 show that, although both methods achieve similar performance on the training set ($\sim$ 96%), MSG outperforms SSG on the validation ($\sim$ 94% vs. $\sim$ 91-92%) and test sets ($\sim$ 90% vs. $\sim$ 84-87%). This demonstrates the increased efficiency of MSG in complex scenarios. However, this performance improvement comes at the cost of significantly higher computation time, as MSG requires the extraction of local features at multiple scales, thus increasing its computational complexity.

4.3 K-means vs. FPS

In PointNet++'s original algorithm, the sampling and grouping phase is performed by applying the Farthest Point Sampling (FPS) method to identify cluster centroids, followed by a ball query to group neighboring points around these centroids. A pertinent question arose as to the choice of the FPS method in this context.

In the paper [17] by Charles R. Qi et al. the authors justify the use of FPS by highlighting its excellent spatial coverage of the point cloud. Indeed, centroids are selected to maximize the distance between them, which favors optimal data dispersion. This ensures that the sampling effectively represents the overall structure of the pointcloud.

To extend this analysis, I wanted to explore how an algorithm like K-means clustering would react when replacing FPS in the PointNet++ model. Unlike FPS, K-means minimizes intra-cluster variance, i.e. it identifies dense clusters of

points in the cloud. This approach could offer an interesting alternative, by concentrating sampling on areas of high density rather than maximum spatial distribution.

This section gives a detailed presentation of the K-Means algorithm, followed by an explanation of its parallel implementation in C++/CUDA. Finally, a comparative analysis is made between sampling and grouping based on K-Means, and those performed using Farthest Point Sampling combined with the Ball Query method.

4.3.1 Description of K-means

K-Means is one of the most popular clustering algorithms. Clustering consists in grouping data with similar features. In our specific case, it involves grouping the points that are spatially closest to each other.

Here are the main steps of the K-Means algorithm for a pointcloud xyz of N points:

1. **K Assignement**: choose the number of clusters K
2. **Initialization**: select K centroids at random among points in xyz
3. **Cluster Assignement**: assign each point of xyz to the closest centroid by calculating euclidean distance between each point and each centroid
4. **Updating**: update centroids position by calculating mean of all data points assigned to each cluster
5. **Repeating**: repeat steps 3 and 4 until convergence

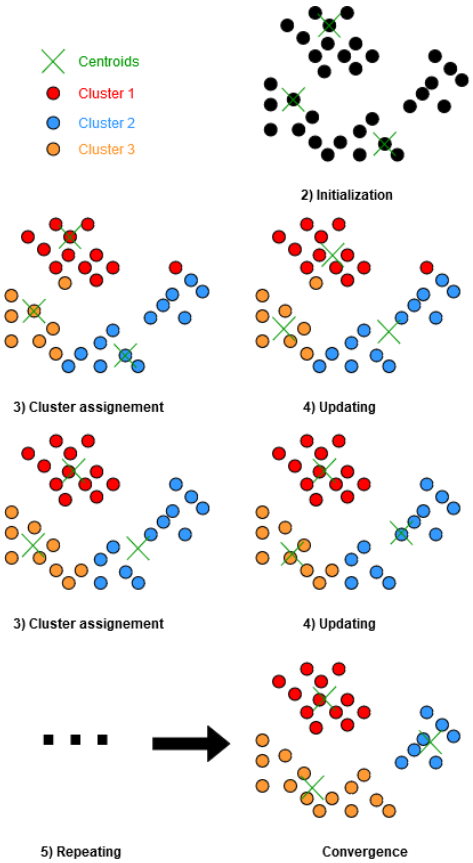


Figure 4.4: K-means clustering

4.3.2 K-means CUDA implementation

To investigate how ModelNet10 classification worked with PointNet++, I used a tensorflow implementation [24] by David Griffith (a research scientist in 3D computer vision and scene understanding). This implementation is based on the original paper [25] by Charles R.Qi, an Ai researcher from Stanford University. In these 2 implementations, the major point cloud processing operations (custom operations and kernels) used for sampling and grouping are designed in C++ and integrated into the tensorflow library via tf.load_op_library. These operations are designed in C++ to be compatible with CUDA (Compute Unified Device Architecture), a technology developed by NVIDIA which uses graphic processor units (GPUs) instead of central processing units (CPUs). The use of these GPUs enables calculations to be carried out in parallel, drastically reducing calculation times for large datasets.

As the sampling and grouping methods are implemented in parallel, I have also implemented the K-Means algorithm in C++/CUDA. Here are the main features of this implementation:

First, let us define the various terms specific to GPU parallelization in CUDA:

- **Threads** are the smallest units of execution in CUDA. Each thread executes a part of the kernel code independently and has a unique identifier in its block (**threadIdx.x**)
- CUDA manages threads into a **thread block** that has a unique identifier **blockIdx.x**
- **gridDim.x** is the number of blocks
- **blockDim.x** is the number of threads per block

Figure 4.5 illustrates the CUDA Architecture for a unique grid of 2 blocks with 4 threads each. With this architecture, 8 calculations can be performed in parallel.

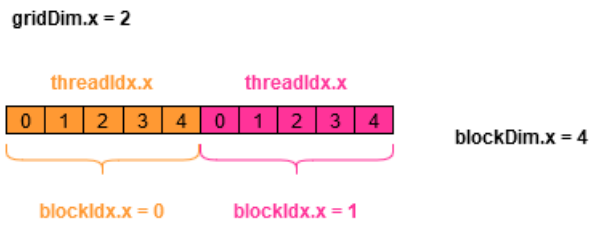


Figure 4.5: CUDA Architecture

Let us use an example of two nested loops in C++ to illustrate how CUDA distributes work between blocks and threads:

```
for (int i=blockIdx.x;i<b;i+=gridDim.x){
    for (int j=threadIdx.x;j<n;j+=blockDim.x){
        float x = dataset[i*n*3 + j*3 + 0];
        float y = dataset[i*n*3 + j*3 + 1];
        float z = dataset[i*n*3 + j*3 + 2];
```

These two loops, which are repeated in my K-means implementation in C++, make it possible to process several point clouds simultaneously and access each of the n points individually.
Let *dataset* be the point cloud coordinates of size $(b \times n \times 3)$ where b is the batch size (number of training examples used in one iteration of model training.), n the number of points in the point cloud and 3 represents the x, y and z coordinates. The vector is flattened into a 1D structure of size $(b \cdot n \cdot 3)$ to facilitate data access in a linear memory layout, ensuring compatibility with CUDA's memory model and enabling efficient indexing.

On Table 4.1, we can see how we can recover the values x,y,z by the repartition of threads and block of CUDA for b=6 pointclouds of n=10 points and gridDim.x=2, blockDim.x=4. In the table, we look at block of index 0 and we see that we can get indices of the xyz coordinates of the 10 points of the first pointcloud by indexation of threads in the block. The same principle applies to the block 1. These two blocks work in parallel, and once the execution of one is complete, index i is incremented to a new value (for example, after the end of execution for i=0, the block with index 0 then processes i=2). Threads work in the same way, first executing calculations for indices marked in blue, then those marked in red and finally those marked in black.

blockIdx.x	i	threadIdx.x	j	i*n*3 +j*3 + 0	i*n*3 +j*3 + 1	i*n*3 +j*3 + 2
0	0,2,4	0	0,4,8	0,12,24	1,13,25	2,14,26
		1	1,5,9	3,15,27	4,16,28	5,17,29
		2	2,6	6,18	7,19	8,20
		3	3,7	9,21	10,22	11,23
1	1,3,5	0	0,4,8			
		1	1,5,9			
		2	2,6			
		3	3,7			

Table 4.1: Example of indexing and distribution of threads and blocks in memory

Algorithm 2 K-means CUDA implementation

CUDA: Calculations of the various functions are carried out using the parallel work of 32 blocks of 512 threads each. All vectors are flattened.
$b \leftarrow$ batch size: CUDA $\longrightarrow$ for (int i=blockIdx.x;i<b;i+=gridDim.x)
$n \leftarrow$ nbr of points: CUDA $\longrightarrow$ for (int j=threadIdx.x;j<n;j+=blockDim.x)
$m \leftarrow$ nbr of clusters: CUDA $\longrightarrow$ for (int k=threadIdx.x;k<m;k+=blockDim.x)
$dataset \leftarrow$ point cloud coordinates $(b \cdot n \cdot 3)$
$centroids \leftarrow$ centroids coordinates $(b \cdot m \cdot 3)$
$clusters \leftarrow$ cluster indexes for each point $(b \cdot n)$
$cluster\_sums \leftarrow$ sum of coordinates of each cluster $(b \cdot m \cdot 3)$
$cluster\_sizes \leftarrow$ size of clusters $(b \cdot m)$
$cluster\_SSE \leftarrow$ Sum of squared error of each cluster $(b \cdot m)$
Let $maxIter \leftarrow$ max number of iterations

1. *kmeans_centroids_Initialization* : initialize *centroids* to the first m points of each object in the batch.
2. While $t < maxIter$
   (a) *kmeans_Cluster_ assignement* :
   - Scans each point in the dataset.
   - For each centroid, calculates the euclidean distance between the point and the centroid.

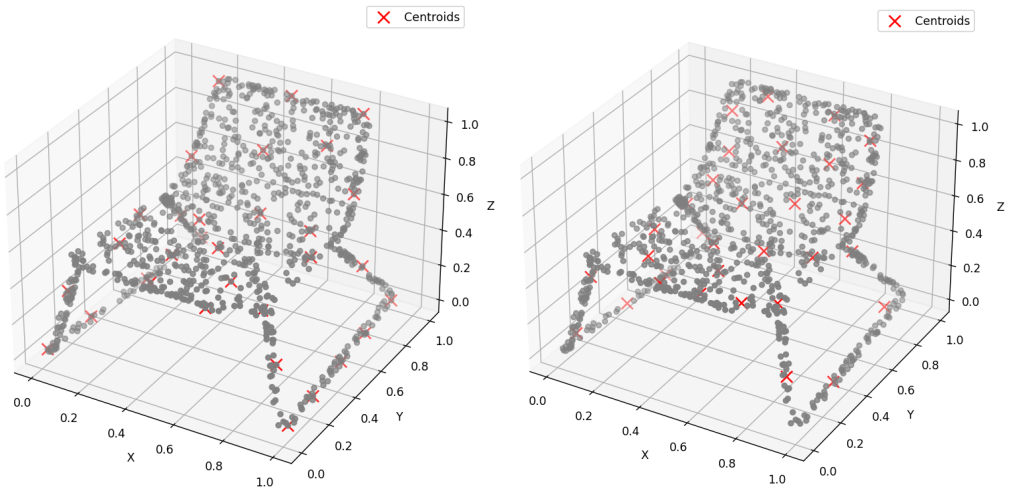
$$distance = (x_c - x)^2 + (y_c - y)^2 + (z_c - z)^2$$

   - Identifies the nearest centroid of each point and updates *clusters*.

   (b) *kmeans_centroids_update*:
   - Resets cluster sums and sizes.
   - Calculates the sums of the coordinates of the points assigned to each cluster, as well as the cluster sizes.
   - In order to prevent a possible empty cluster, calculates the Sum of squared errors (SSE) of each cluster to measure its dispersion. The larger the SSE, the more is the cluster dispersed.
   - Identifies empty clusters and reassigns the centroid for each, choosing the point that contributes most to the SSE of the most dispersed cluster.
   - Recalculates centroid position (weighted average)

$$centroids[i * m * 3 + k * 3 + 0] = \frac{cluster\_sums[i * m * 3 + k * 3 + 0]}{cluster\_sizes[i * m + k]}$$

4.3.3 Replacing FPS by K-means for sampling



FPS K-means

Figure 4.6: Sampling phase comparison between FPS and K-means

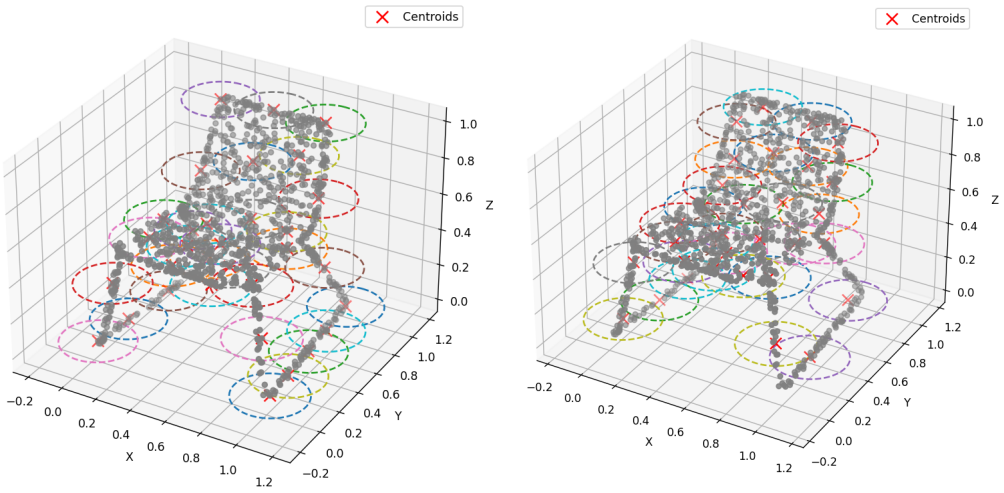
We looked at the impact of using the K-Means algorithm to replace Farthest Point Sampling (FPS) for the sampling phase, followed by a ball query local search method.

Figure 4.6 illustrates the differences between FPS and K-Means for the selection of 32 distinct centroids for a "chair" class object. Two major observations emerge from this comparison:

- **Dependence on initial centroids**: Unlike FPS, which generates systematically deterministic results, K-Means is sensitive to initial centroids choice, which can lead to variations in results.
- **Spatial coverage**: FPS guarantees a balanced and homogeneous distribution of centroids in the point cloud, whereas K-Means can produce a less uniform coverage (as can be seen in the leg of the chair in the bottom right-hand corner on above Figure). The centroids selected by FPS capture the shape of the object more faithfully, as they are mainly located at the borders, whereas those generated by K-Means represent an average of the surrounding points, which can attenuate contour details.

Based on these observations, we expect FPS to offer better classification performance. However, the results on ModelNet10 show that classification performance

is of similar order for both methods. This can be explained by the high number of centroids used in the successive sampling phases (512 in the first phase and 128 in the second), which reduces the differences in coordinates between the centroids calculated by the two methods. In fact, in relatively well sampled clouds such as ModelNet10, the points chosen by FPS and K-means are likely to cover the same important areas. All the more so as the size of the ball (as can be seen on Figure 4.7) used in the grouping phase is relatively large, allowing all the complex information to be extracted from the point cloud, whatever the sampling method. However, if we were to perform the test on non-uniform density datasets as can be found in real datasets, it is possible that K-means would only capture information from the densest regions.



FPS K-means

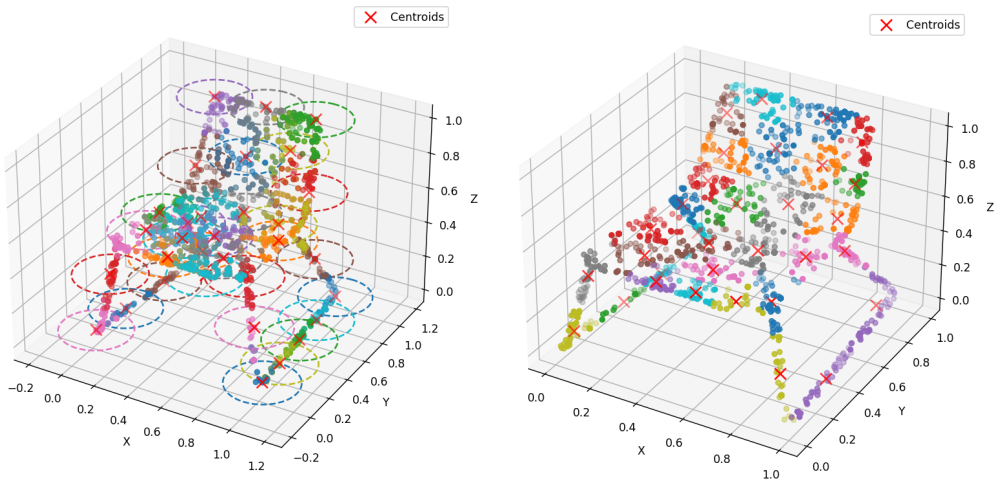
Figure 4.7: Grouping phase by ball query around centroids

4.3.4 K-means for sampling and grouping

A second interesting analysis consists in examining the performance obtained when using K-means for its primary function, namely clustering. For this, PointNet++'s sampling and grouping steps were replaced by the K-means clustering algorithm.

The ball query method offers a significant advantage over K-means. Ball Query covers more points than K-means clustering, because it allows overlaps between clusters. This means that several points can belong to different clusters, which is particularly useful in high-density areas such as object corners. Conversely, K-

means assigns each point to a single cluster, which limits the wealth of information captured in these dense regions.



FPS and Ball query K-means clustering

Figure 4.8: Sampling and Grouping phase comparison between FPS followed by ball query and K-means clustering

Furthermore, in the second sampling phase of PointNet++, the sphere radius increases (from 0.2 to 0.4), enabling Ball Query to capture an even greater number of neighbors and have more overlaps. This ability to capture more information is directly reflected in the classification results, confirming our expectations.

With K-means, classification performance is clearly inferior and shows considerable variability (from 65% to 80 % in classification accuracy depending on the number of clusters and maximum points per cluster). In particular, K-means has trouble distinguishing between similar, dense classes such as desk, dresser and nightstand. These objects, often composed of square blocks with a high density of points in the center, highlight K-means' difficulty in capturing sufficient information in dense regions.

To illustrate this difference, let us take the example of the intersection between the left rear leg, the seat, and the back of the chair in the above Figure. With Ball Query, the orange, brown and khaki green clusters contain information on this crucial corner of the object's structure. In contrast, with K-means, only one cluster (the pink one) captures this information, reducing the granularity and richness of the extracted data.

It is also important to note that this example only shows 32 clusters, whereas in PointNet++, we will have 512 centroids the first time we perform the sampling

and grouping step. This means that, in the case of Ball Query, these critical points could be included in around 30 clusters at that time. With K-means, on the other hand, the average number of points per cluster is much lower, so the algorithm captures even less important information.

This limitation of K-means is linked to its difficulty in capturing information as rich as that obtained with Ball Query. Indeed, Ball Query allows spheres to be superimposed and common information to be included across multiple clusters, which is essential for good abstraction.

Although it is possible to obtain acceptable results with K-means by carefully optimizing parameters (such as the number of clusters or abstraction layers), the algorithm remains more complex and less robust than the Farthest Point Sampling (FPS) approach followed by Ball Query. Consequently, there is no convincing reason, in this context, to replace FPS and Ball Query with K-means for sampling and clustering. The authors' choice of PointNet ++ therefore seems well justified.

4.4 Classification results

MNIST digit classification

Method	Error rate (%)
Multi-layer perceptron	1.60
LeNet5	0.80
Network in Network	**0.47**
PointNet (vanilla)	1.30
PointNet	0.78
PointNet++	0.51

ModelNet40 shape classification

Method	Input	Accuracy (%)
Subvolume	vox	89.2
MVCNN	img	90.1
PointNet (vanilla)	pc	87.2
PointNet	pc	89.2
PointNet++	pc	90.7
PointNet++(with normal)	pc	**91.9**

Table 4.2: Point Set Classification results comparison with state of art methods [17]

In 2017, PointNet++ achieves state-of-the-art performance on complex 3D point cloud benchmarks, and has made it possible to integrate local relationships between points, setting it apart from its predecessor PointNet. This ability to capture local structures gives it a significant advantage in terms of accuracy, particularly in classification tasks. However, a major limitation of PointNet++ lies in its high computational cost, due to the sampling and grouping layers, as well as the multi-scale grouping process. The various methods derived from PointNet ++ such as Pointweb or RandLA-Net have subsequently made it possible to reduce the inference time of the method while retaining its accuracy.

Chapter 5

Comparison of deep learning and traditional methods

In this section, we first describe the different classification results obtained using Point Feature Histograms in comparison with PointNet. The idea here is to compare a traditional method with the deeep learning method, which is the reference model that defined the theoretical and methodological foundations of direct point cloud processing, inspiring a majority of subsequent models. Finally, we describe the considerable improvement PointNet++ has brought over PointNet.

5.1 Traditional method: Point Feature Histogram

5.1.1 SVM results for local and global features

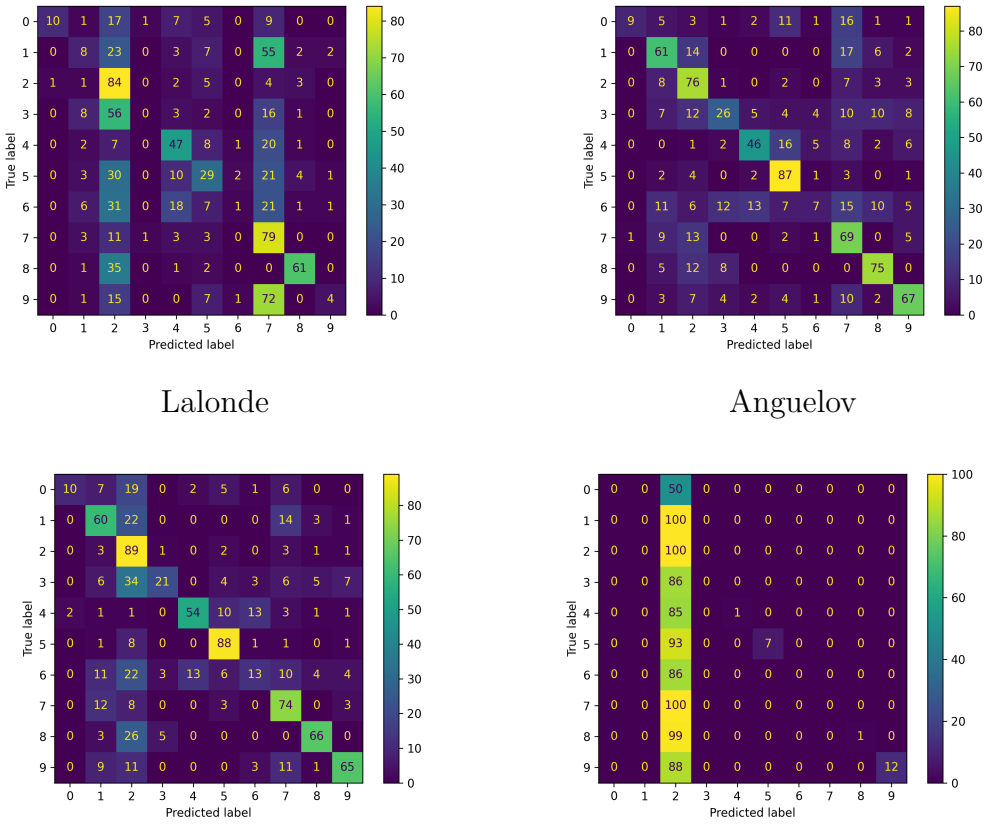
Figure 5.1 shows several confusion matrices representing the classification of the SVM according to the features used. If we use Lalonde alone, we obtain a test accuracy of 35.57%, with Anguelov features we obtain 57.59% and if we combine the 2 local features we slightly improve the accuracy, reaching 59.47%. However, if we add the global volume variable, we see that the model becomes very inefficient, with an accuracy of 13.32%.

We can conclude that the model undergoes over-fitting when the global volume variable is added. In fact, what happens is that the volume value itself determines a particular object. However, it is not capable of distinguishing between different classes.

We can also see that Anguelov's features seem more relevant than Lalonde's for ModelNet10 classification. However, we do obtain an improvement in performance

when we combine the two features.

As the results by adding global value are really bad, all next results of Point feature histograms, used for comparison with PointNet, are calculated with the local features (Anguelov and Lalonde) only as these combination of features seem to perform better for classification on ModelNet10 dataset.



Lalonde

Anguelov

Local features (Lalonde + Anguelov)

Global (volume) + local features

Figure 5.1: Confusion matrix results for PFH with Lalonde features, Anguelov features, local features and local + global features using the complete ModelNet10 dataset with 512 sampled points

5.1.2 Summary of Lalonde results

In the figure below, the impact of the number of classes on SVM classification with only Lalonde features is clearly visible: an increase in the number of classes

leads to a decrease in accuracy. On the other hand, sampling a greater number of points provides more information, which improves the model's ability to distinguish between different classes, as expected.

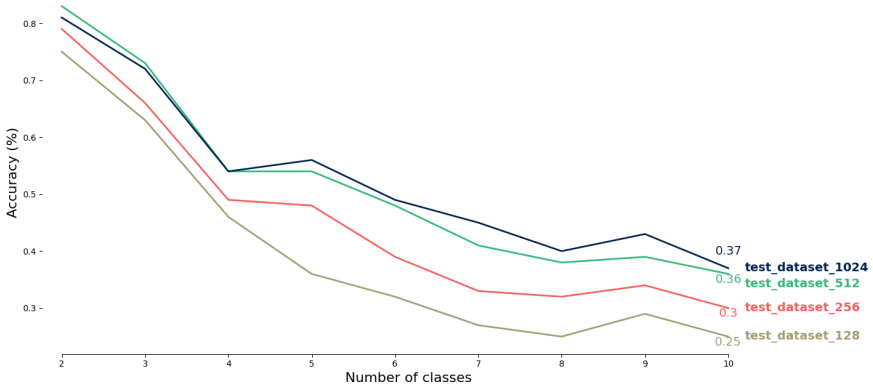


Figure 5.2: SVM Classification with feature vector = Lalonde features

5.1.3 Summary of Lalonde + Anguelov results

The decrease observed here is less marked than that observed with only Lalonde features for point clouds containing between 128 and 512 sampled points. On the other hand, a significant drop in accuracy is observed for the dataset containing 1024 sampled points, the origin of which will be analyzed in greater detail later in this chapter.

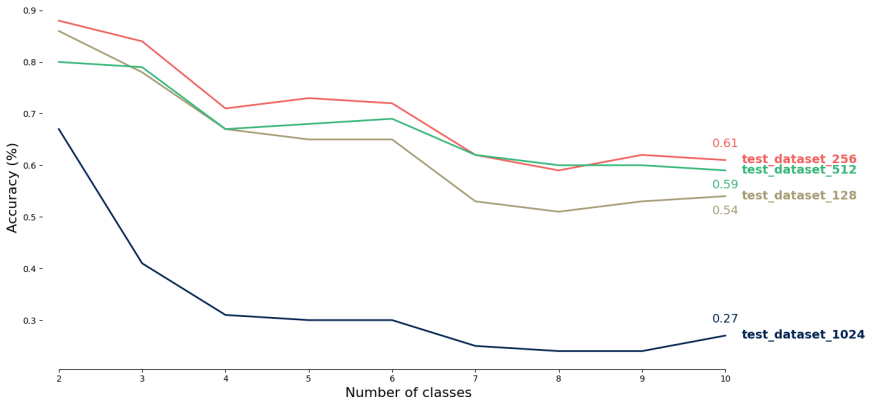


Figure 5.3: SVM Classification with feature vector = Lalonde and Anguelov features

5.2 Deep learning: PointNet

5.2.1 Evolution of loss and accuracy

Figures 5.4 and 5.5 show the evolution of loss and accuracy trained on ModelNet with 10 classes and 1024 sampled points. We can see that the validation accuracy curve closely follows the training curve, just as the validation loss curve follows the training loss curve. The model seems to converge in a fairly stable way, but when we observe the test accuracy, there is a significant decrease in accuracy on the test, which is 0.82, while the final accuracy values for training and validation are 0.96 and 0.92 respectively. This indicates a slight overfitting, which could be attributed to several factors, such as too high number of epochs, excessive model complexity relative to the dataset size or simply an inappropriate choice of hyper-parameters. Further examples of the evolution of loss and accuracy over time can be found in Appendix B.

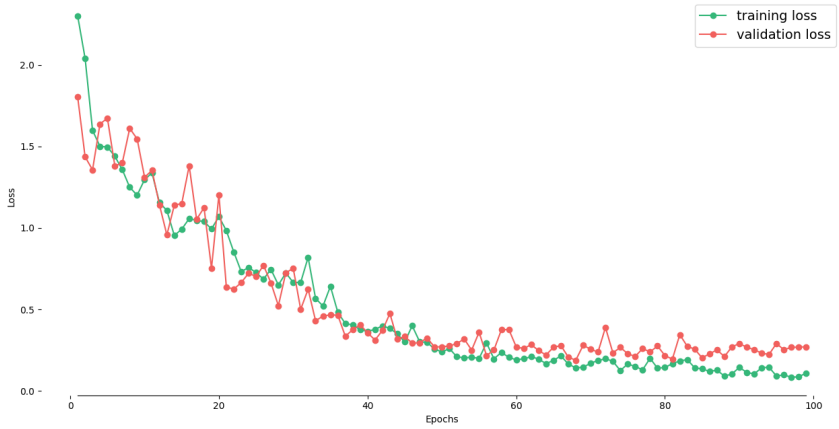


Figure 5.4: Evolution of loss over time for ModelNet with 10 classes and 1024 sampled points during fitting

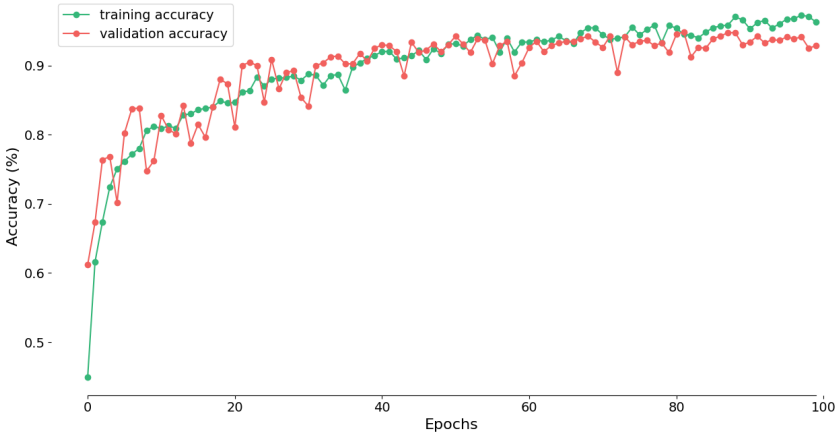


Figure 5.5: Evolution of accuracy over time for ModelNet with 10 classes and 1024 sampled points during fitting

5.2.2 Summary of PointNet results

This last Figure (5.6) shows a summary of all test accuracy of classes from 2 to 10 and test dataset with 128 to 1024 sampled points. The final observed values for 10 classes (84%,83%,85%,82%) are much higher than those observed with Point Feature Histogram in figure 5.3 (54%,61%,59%,27%).

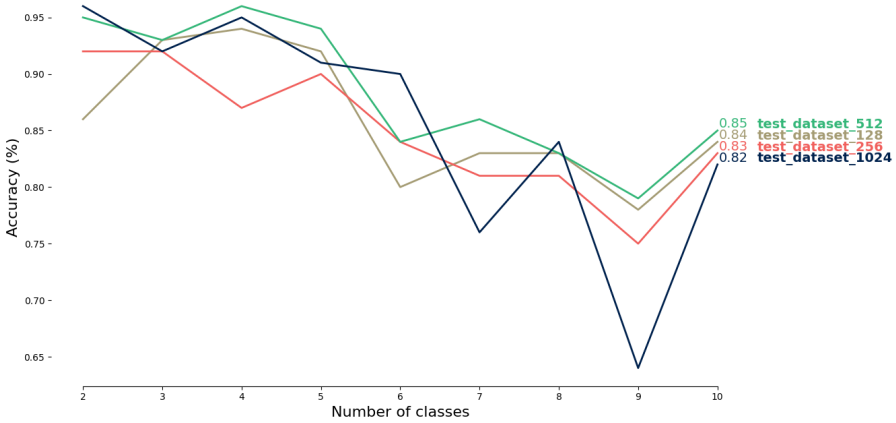


Figure 5.6: PointNet classification results

5.3 Comparison of PFH and PointNet

5.3.1 Expected Results

Inference time

What I was expecting to observe is a much shorter inference time for PointNet because PointNet is a neural network and, once trained, the point cloud is processed directly. The only operations performed through the neural network are only matrix mutliplications and non-linear activation function operations.

In contrast, for the traditional method, for each sample and each point in that sample, we need to define its neighborhood and recalculate the local features before transmitting the final feature vector to the SVM classifier.

With PointNet, as the number of points increases, so does the number of calculations required, as each point must pass through the MLPs. However, these calculations can be performed in parallel with an optimized processing unit such as a GPU, as the calculations are independent for each point.

For Point Feature Histograms, the impact is much greater, as the calculations are performed sequentially in my implementation.
Initially, when implementing the method, I had not considered developing a parallel version of PFH. However, after implementing K-means in CUDA, it appeared quite feasible to design a GPU version of PFH. The inference time results obtained in next session must therefore be interpreted in the light of the fact that PointNet was run on a GPU, whereas PFH was run on a CPU.

Accuracy

I also expect to get better results in terms of accuracy with PointNet, as PointNet processes a point cloud as a whole. Point Feature Histograms, on the other hand, are based on hand-crafted features and will undoubtedly have difficulty classifying complex objects.

5.3.2 Effective Results

Inference time

Tables 5.1, 5.2 and 5.3 present the results of inference time for PointNet and Point Feature Histograms.

Below are defined the key data to understand the tables presented in this section:

- The total inference time is the total time the model takes to calculate the prediction of the test set.

- The support is the number of samples of the specific dataset.

- The inference time per sample is the division of the total inference time by the support of the test dataset.

Table 5.1: Inference time for Point Feature Histogram wrt number of classes with 512 sampled points

Classes	Total inference time [s]	Support	Inference time per sample [ms/sample]
2 classes	9.596	150	63.97
4 classes	37.704	336	112,21
6 classes	39.519	522	75.71
8 classes	73.963	708	104.47
10 classes	67.621	908	74.47

Table 5.2: Inference time for Point Feature Histogram wrt number of sampled points for test dataset with 6 classes

Points	Total inference time [s]	Support	Inference time per sample [ms/sample]
128	9.415	522	18.04
256	13.227	522	25.34
512	39.519	522	74.47
1024	158.015	522	302.71

Table 5.1 shows that the total inference time increases as the number of classes increases. More precisely, as the number of samples increases, the total time inference increases. Therefore, inference time per sample stays around 100ms per sample. Indeed, what takes time for prediction with the PFH method is to calculate the feature histograms (before giving it to SVM).

On Table 5.2, we can see that inference time is growing exponentially the higher the number of points. That number explodes with 1024 points (goes from 39 s with 512 points to 158 s with 1024 points and from 74.47 ms per sample to 302.71

ms for total inference time and inference time per sample respectively). The more points in the point cloud, the more neighbors to take into account in the feature construction phase, the more calculations.

Table 5.3: Inference time for PointNet wrt number of classes with 512 sampled points

Classes	Total inference time [s]	Support	Inference time per sample [ms/sample]
2 classes	5.126	150	34.17
4 classes	5.14	336	15.30
6 classes	5.126	522	9.8
8 classes	5.127	708	7.24
10 classes	5.129	908	5.6

For PointNet however, the total inference time stays constant no matter the number of samples and the number of points. As a result, inference time per sample decreases drastically as the support increases: with a support of 150 the inference sample time for PointNet is ≈ 2 times lower than PFH (34ms for PointNet and 64 ms for PFH) and more than 10 times lower with a support of 908 (5.6 ms for PointNet and 74,5 ms for PFH). Batch processing and parallelism in the deep neural model are responsible for the constant total inference time in PointNet.

All these results are consistent with the expected results presented in the previous section.
One solution to reduce inference time for PFH is to limit the number of neighbors for which local features are calculated. However, the accuracy might decrease if the number of neighbors is not high enough.

Accuracy

In this section, the performance of our two models will be compared.

The first two graphs (Figures 5.7 and 5.8) represent the impact of the number of sampled points in the point cloud with respect to the accuracy of the model.
For Point feature histogram, it is clear on the figure 5.7 that the observed accuracies with 1024 points are very low compared to others. One might think that a large number of points would provide more information, thus improving classification, but in this case there is probably an excess of points leading to over-fitting and thus reducing performance.

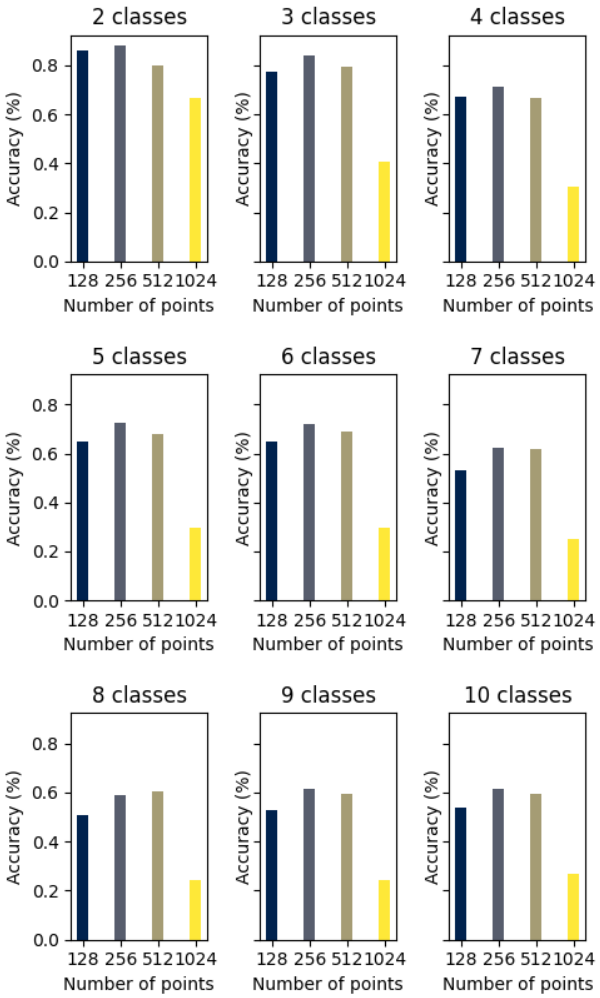


Figure 5.7: Impact of number of points for Point Feature Histogram classification

For PointNet however, we can rather observe almost same performance with respect to the number of points. But if we look carefully at specific datasets like 7 classes dataset or 9 classes dataset, the accuracy of test_dataset_1024 has a significantly drop in performance. This drop suggests also over-fitting as mentioned in section 5.2 or eventually under-fitting if the model had not have the time to converge.

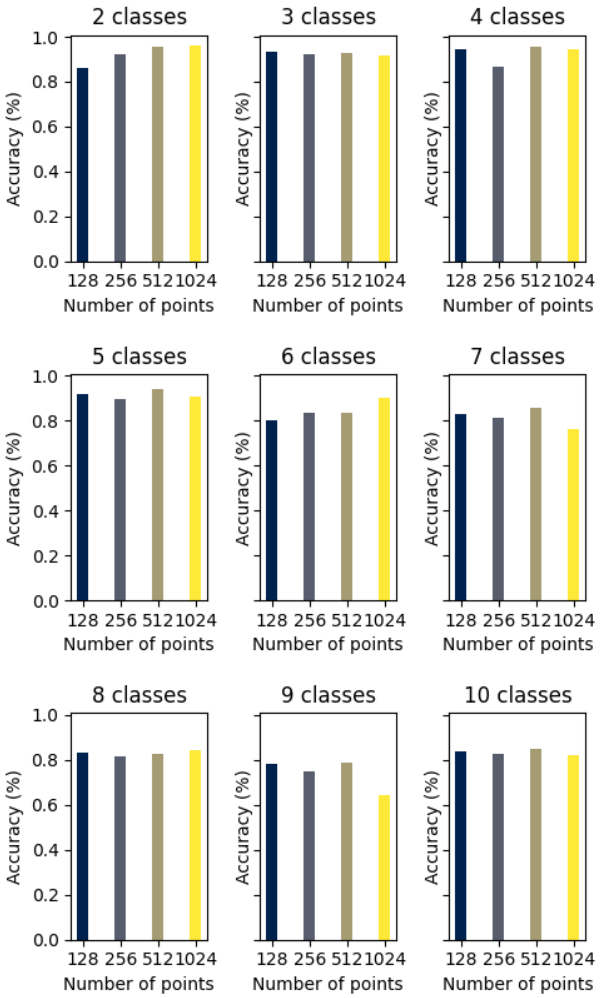


Figure 5.8: Impact of number of points for PointNet classification

In the next graphs (Figure 5.9), we will compare directly the accuracies of PointNet and PFH.

On Figure 5.9, we can observed that PFH manages to approach PointNet's accuracy when the number of classes is limited. However, as the number of classes increases, PFH's accuracy falls considerably in comparison with PointNet's accuracy. In addition, the gap between PFH and PointNet's performance widens as the number of points increases, with PFH performing increasingly worse than PointNet.

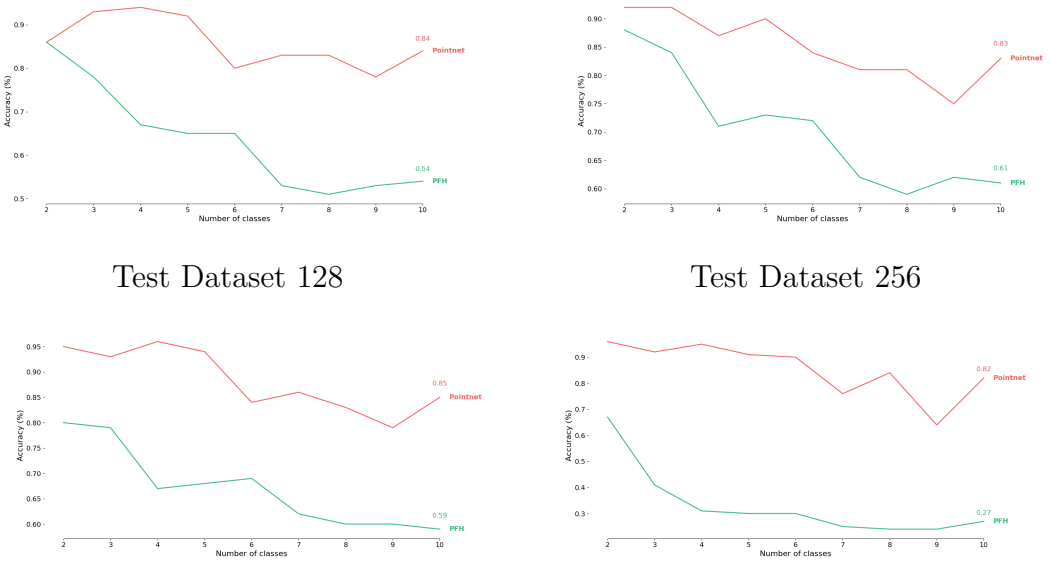


Figure 5.9: PointNet vs. Point Feature Histogram (PFH) accuracy with respect to number of sampled points

The Figure 5.10 gives more details on 3 specific cases:

1. Dataset with 2 classes and a small number of points (128): PFH perfoms almost as well as PointNet. However we much prefer the recall and precision values of PointNet:
   - For PointNet, the precision is balanced between the 2 classes (87% for both) and the recall is good, especially for the class 1 (95%)
   - For PFH, the precision of class 1 is perfect but it is at the cost of a very low recall (58%). This means that we will have many false negatives and in most applications of real life it is something we want to avoid.
2. Dataset with 6 classes and a mean number of points (256): all F1-score of PointNet are greater than those of PFH. The PFH accuracy is still acceptable but there is already a clear gap between PFH and PointNet (72% against 83%).
3. Dataset with 10 classes and a higher number of points (1024): PFH performs really bad (27%) whereas PointNet performance decreases a bit with a higher number of points and classes but stay stable (82%).

	Pointnet 128	precision	recall	f1-score	support
Bathtub	0	0,87804878	0,72	0,791208791	50
Bed	1	0,871559633	0,95	0,909090909	100
	accuracy	0,873333333	0,873333333	0,873333333	0,873333333
	macro avg	0,874804207	0,835	0,85014985	150
	weighted avg	0,873722682	0,873333333	0,86979687	150

	PFH 128	precision	recall	f1-score	support
Bathtub	0	1	0,58	0,734177215	50
Bed	1	0,826446281	1	0,904977376	100
	accuracy	0,86	0,86	0,86	0,86
	macro avg	0,91322314	0,79	0,819577295	150
	weighted avg	0,884297521	0,86	0,848043989	150

	Pointnet 256	precision	recall	f1-score	support
Bathtub	0	1	0,74	0,850574713	50
Bed	1	0,772357724	0,95	0,852017937	100
Chair	2	0,956521739	0,88	0,916666667	100
Desk	3	0,867924528	0,534883721	0,661870504	86
Dresser	4	0,640625	0,953488372	0,76635514	86
Monitor	5	1	0,89	0,941798942	100
	accuracy	0,837164751	0,837164751	0,837164751	0,837164751
	macro avg	0,872904832	0,824728682	0,831547317	522
	weighted avg	0,867094264	0,837164751	0,836023938	522

	PFH 256	precision	recall	f1-score	support
Bathtub	0	0,869565217	0,4	0,547945205	50
Bed	1	0,702479339	0,85	0,769230769	100
Chair	2	0,594936709	0,94	0,728682171	100
Desk	3	0,88372093	0,441860465	0,589147287	86
Dresser	4	0,820895522	0,639534884	0,718954248	86
Monitor	5	0,763636364	0,84	0,8	100
	accuracy	0,720306513	0,720306513	0,720306513	0,720306513
	macro avg	0,772539014	0,685232558	0,692326613	522
	weighted avg	0,758966508	0,720306513	0,708209361	522

	Pointnet 1024	precision	recall	f1-score	support
Bathtub	0	0,971428571	0,68	0,8	50
Bed	1	0,871559633	0,95	0,909090909	100
Chair	2	0,930693069	0,94	0,935323383	100
Desk	3	0,87037037	0,546511628	0,671428571	86
Dresser	4	0,70212766	0,38372093	0,496240602	86
Monitor	5	0,989583333	0,95	0,969387755	100
Nightstand	6	0,48	0,697674419	0,568720379	86
Sofa	7	0,816666667	0,98	0,890909091	100
Table	8	0,777777778	0,91	0,838709677	100
Toilet	9	0,932692308	0,97	0,950980392	100
	accuracy	0,81938326	0,81938326	0,81938326	0,81938326
	macro avg	0,834289939	0,800790698	0,803079076	908
	weighted avg	0,83368231	0,81938326	0,813622932	908

	PFH 1024	precision	recall	f1-score	support
Bathtub	0	0	0	0	50
Bed	1	1	0,03	0,058252427	100
Chair	2	0,134770889	1	0,237529691	100
Desk	3	0	0	0	86
Dresser	4	0,810810811	0,348837209	0,487804878	86
Monitor	5	0,931034483	0,27	0,418604651	100
Nightstand	6	0	0	0	86
Sofa	7	0,755555556	0,34	0,468965517	100
Table	8	1	0,26	0,412698413	100
Toilet	9	1	0,25	0,4	100
	accuracy	0,269823789	0,269823789	0,269823789	0,269823789
	macro avg	0,563217174	0,249883721	0,248385558	908
	weighted avg	0,607781743	0,269823789	0,266031156	908

Figure 5.10: Specific comparison of PointNet and PFH accuracy

Conclusion

For point cloud classification, the PointNet method outperforms the Point Feature Histograms method overall. In fact, its performance remains high despite the number of points or the difficulty of separating more classes. The traditional PFH method, on the other hand, seems unable to cope with the increasing complexity of the data. It performs well when only a few classes need to be classified, but drops when the classification task becomes more complex.

5.4 PointNet ++

In this section, we evaluate the performance of the different PointNet++ variants (SSG and MSG) and compare them with those obtained using PointNet. The results are calculated on complete point clouds comprising 10 classes and 1024 points, in line with the methodology established in the previous sections.

Here is how the classification accuracy of PointNet++ performs:

- **PointNet++ SSG** (Single Scale Grouping) achieved accuracy performances in the order of 96-97% for the training set, 91-92% for the validation set and 84-87% for the test set.
- **PointNet++ MSG** (Multi Scale Grouping) achieved slightly different performances: 96% for the training set, 94% for the validation set and 90% for the test set.

MSG is clearly more efficient and robust, thanks to its ability to extract local features at multiple scales. However, this robustness comes at a cost: computation time is considerably longer, as it requires additional calculations to integrate multi-scale information.

PointNet achieves an accuracy of around 82% on the test set. However, its improved version, PointNet++, significantly exceeds this result, increasing accuracy by 8% on the same data set. This improvement is remarkable, especially in a context where modern deep learning models often struggle to achieve even a gain of a few tenths of a percentage. It is important to point out that I have not tested all the possible hyper-parameter combinations on PointNet and PointNet ++, so this difference is probably smaller (as observed on ModelNet40 in section 4.4, where the difference between PointNet and PointNet ++ is more like 3%).

Conclusion

Point cloud processing is starting to become omnipresent in our society for many applications. Today, the majority of point cloud processing methods are based on deep learning. However, these neural networks are often regarded as 'black boxes'. As a matter of fact it is extremely difficult to understand exactly how these models make their decisions, given that they can contain millions or even billions of parameters. As a result many people have a feeling of mistrust about deep learning, which is totally understandable. In their opinion, deep learning often simply means creating a network, experimenting with a multitude of parameters, and then hoping to obtain good results without really understanding the underlying process.

In view of the above, I was eager to try and understand how traditional methods worked before and to be able to interpret the different features extracted from my point cloud myself. I started with exploring the state of the art and looked at the different 3D datasets available and the different tasks to be performed on the dataset. I finally decided to focus on point cloud classification with the ModelNet10 dataset. I chose classification because it is the essential basis for more complex tasks such as segmentation for example.

I continued with a complete analysis of a traditional point cloud processing method: Point Feature Histogram. This method helped me to understand the use of a feature selection algorithm based on margin (SIMBA) and to visualise and interpret local features directly.

The next step was to compare all classification results on ModelNet10 of this traditional method with the reference deep learning point cloud method: PointNet.

After analysis of the results, I can definitely conclude that the PointNet method demonstrates superior performance in terms of accuracy and calculation time in point cloud classification compared to the Point Feature Histograms (PFH) method. PointNet consistently achieves high accuracy, even as the number of points or the number of classes in the dataset increases, all with limited computing time. This

robustness highlights PointNet's ability to handle complex and challenging classification scenarios effectively. In contrast, the traditional PFH method struggles as the complexity of the data grows.

However, PointNet also have its limitations: although it excels at global point cloud representation, it has difficulty extracting finer-grained patterns. With this in mind, PointNet++ was developed and successfully exploited the computing power offered by efficient parallelization and GPU utilization, typical of deep learning networks, while preserving PointNet's direct approach to point cloud processing and taking full advantage of local point cloud structures by grouping points in overlapping local regions. Ultimately, PointNet++ can be seen as an approach combining the principles of PointNet with the exploitation of local representations characteristic of traditional methods.

With PointNet++, we were also able to compare different sampling and grouping algorithms, and conclude that Farthest Point Sampling and ball query are simpler algorithms and better suited to classification in various variable density scenarios, compared to K-means.

Finally, the interior of a deep neural network is likely to remain a complex and mysterious domain, but to improve the model, PointNet++ proved us that is not enough to play Russian roulette in the choice of parameters but it is essential to use our knowledge of geometry, mathematics and even visual analysis, to guide model tuning in a more informed and structured way.

Appendix A

Simba accuracy check additional graphs (Sec 2.4.3)

A.1 Bathub-bed Dataset 256

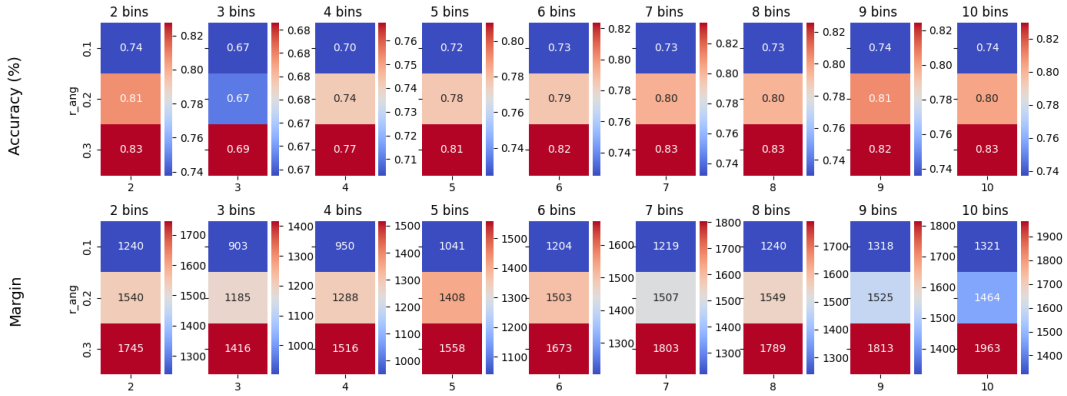


Figure A.1: Comparison of margin and accuracy wrt to the number of bins for choice of the r_ang parameter of Anguelov with the dataset bathtub-bed with 256 sampled points

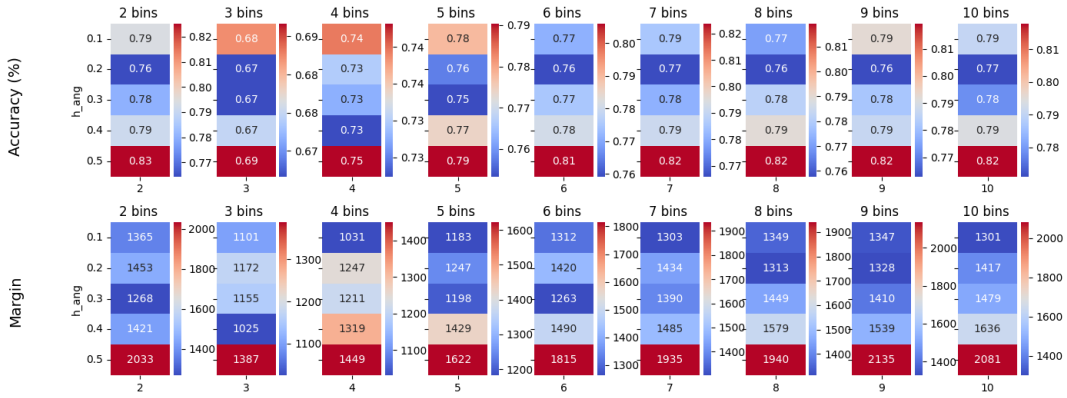


Figure A.2: Comparison of margin and accuracy wrt to the number of bins for choice of the h_ang parameter of Anguelov with the dataset bathtub-bed with 256 sampled points

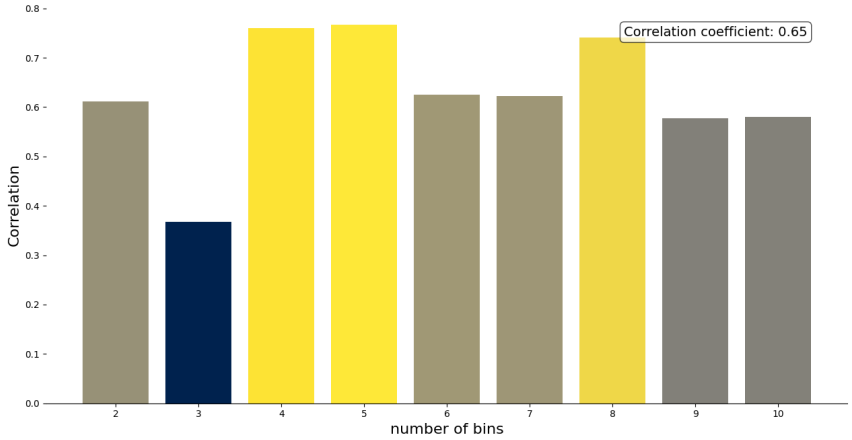


Figure A.3: Correlation between accuracy and margin as a function of the number of bins for bathtub-bed dataset with 256 sampled points

A.2 Bathub-bed-chair-desk Dataset 256

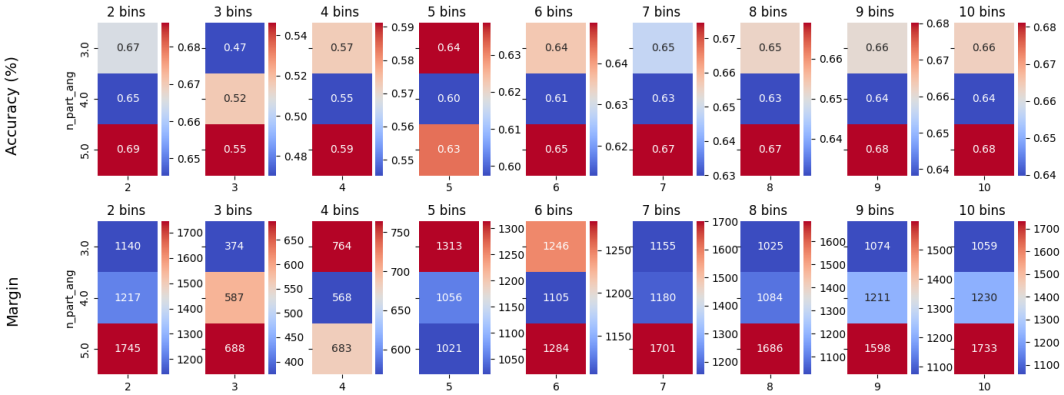


Figure A.4: Comparison of margin and accuracy wrt to the number of bins for choice of the n_part parameter of Anguelov with the dataset bathtub-bed-chair-desk with 256 sampled points

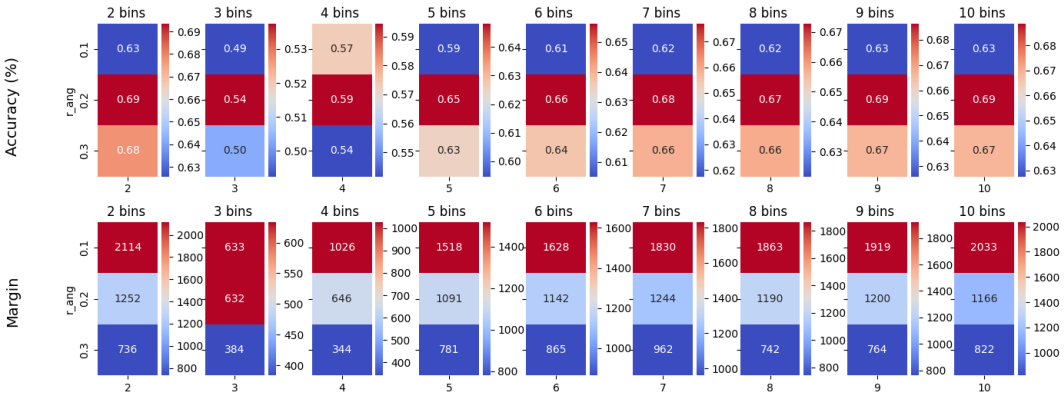


Figure A.5: Comparison of margin and accuracy wrt to the number of bins for choice of the r_ang parameter of Anguelov with the dataset bathtub-bed-chair-desk with 256 sampled points

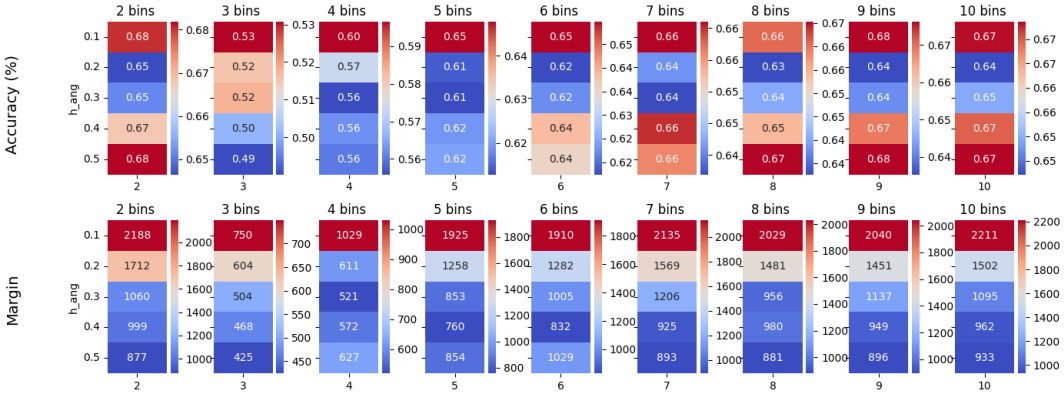


Figure A.6: Comparison of margin and accuracy wrt to the number of bins for choice of the h_ang parameter of Anguelov with the dataset bathtub-bed-chair-desk with 256 sampled points

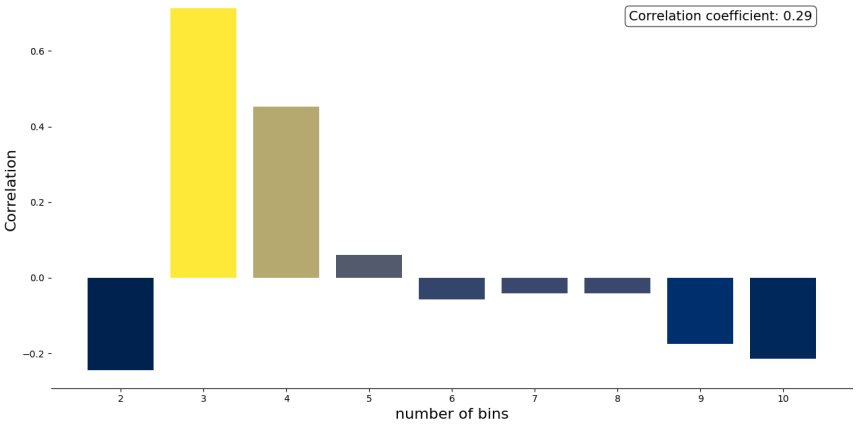


Figure A.7: Correlation between accuracy and margin as a function of the number of bins for bathtub-bed-chair-desk dataset with 256 sampled points

A.3 Bathub-bed-chair-desk Dataset 512

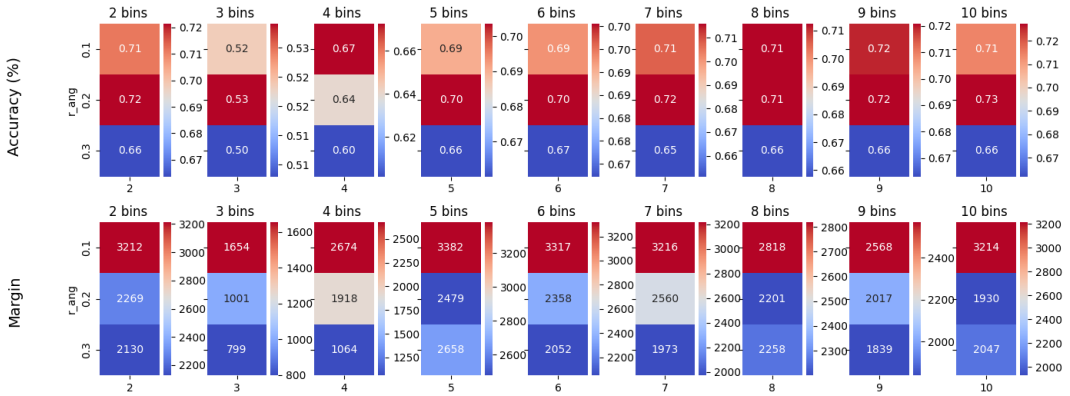


Figure A.8: Comparison of margin and accuracy wrt to the number of bins for choice of the r_ang parameter of Anguelov with the dataset bathtub-bed-chair-desk with 512 sampled points

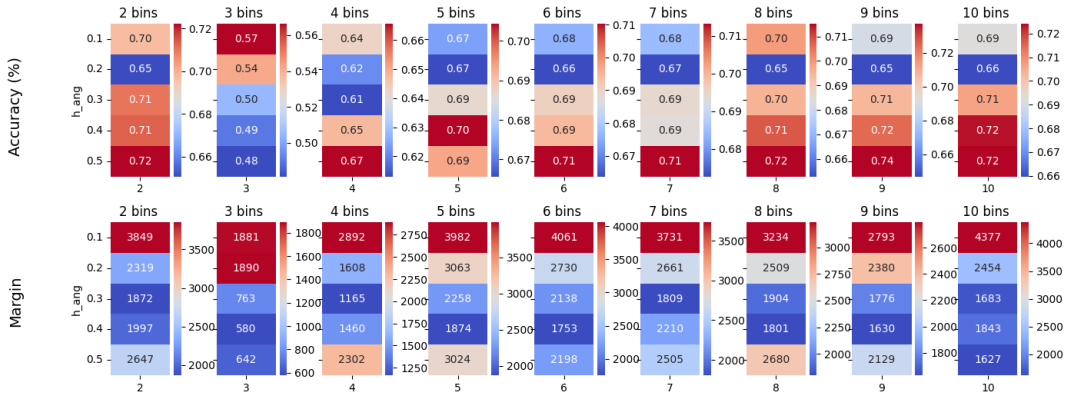


Figure A.9: Comparison of margin and accuracy wrt to the number of bins for choice of the h_ang parameter of Anguelov with the dataset bathtub-bed-chair-desk with 512 sampled points

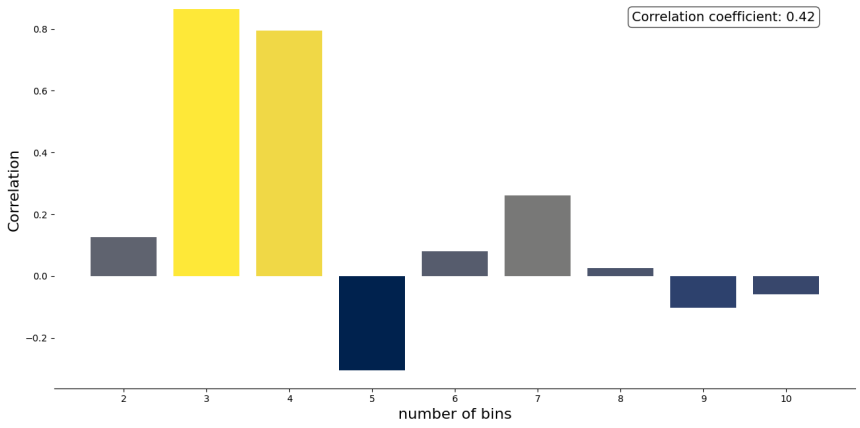


Figure A.10: Correlation between accuracy and margin as a function of the number of bins for bathtub-bed-chair-desk dataset with 512 sampled points

Appendix B

PointNet graphs of loss and accuracy evolution during training phase (Sec 5.2)

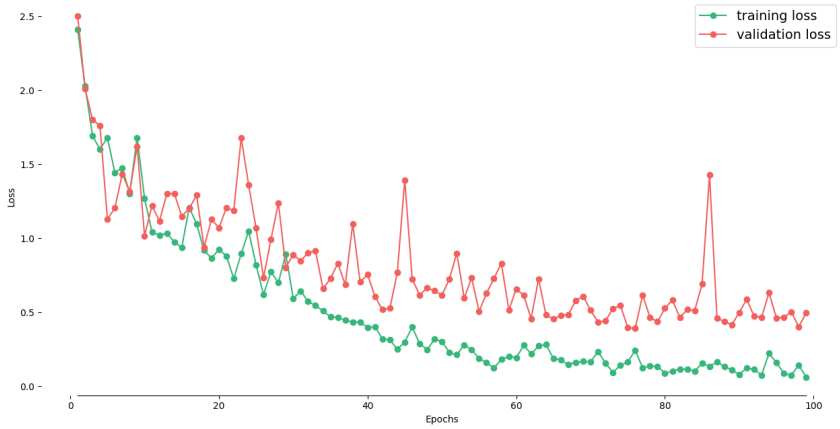


Figure B.1: Evolution of loss over time for Modelnet with 10 classes and 128 sampled points during fitting

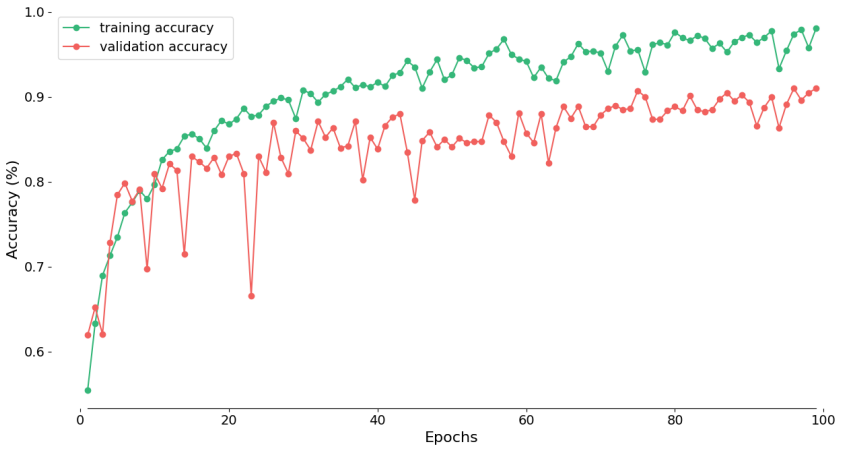


Figure B.2: Evolution of accuracy over time for Modelnet with 10 classes and 128 sampled points during fitting

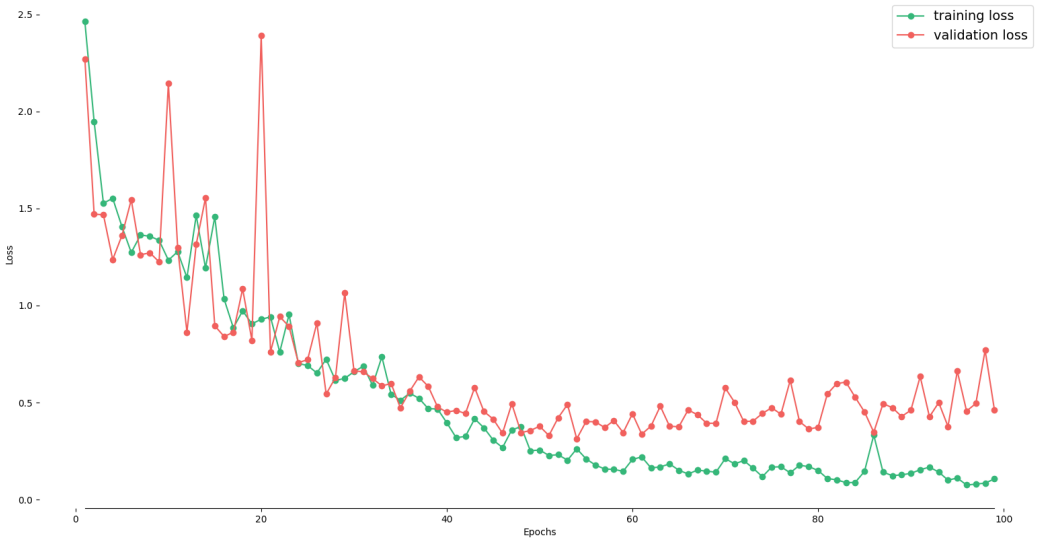


Figure B.3: Evolution of loss over time for Modelnet with 10 classes and 256 sampled points during fitting

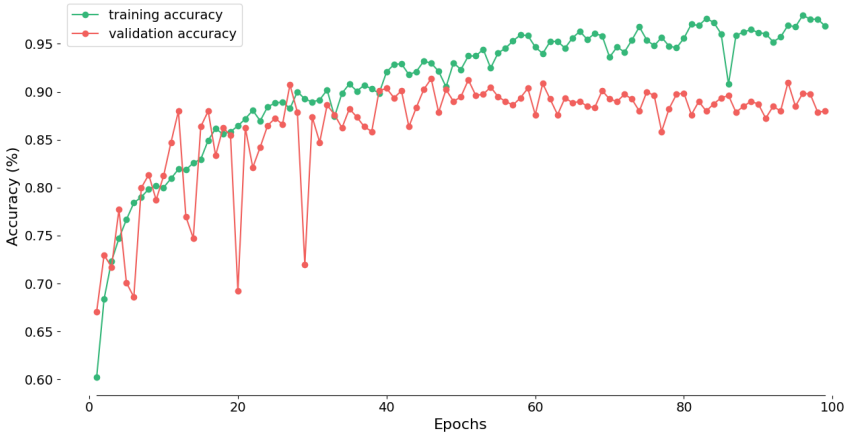


Figure B.4: Evolution of accuracy over time for Modelnet with 10 classes and 256 sampled points during fitting

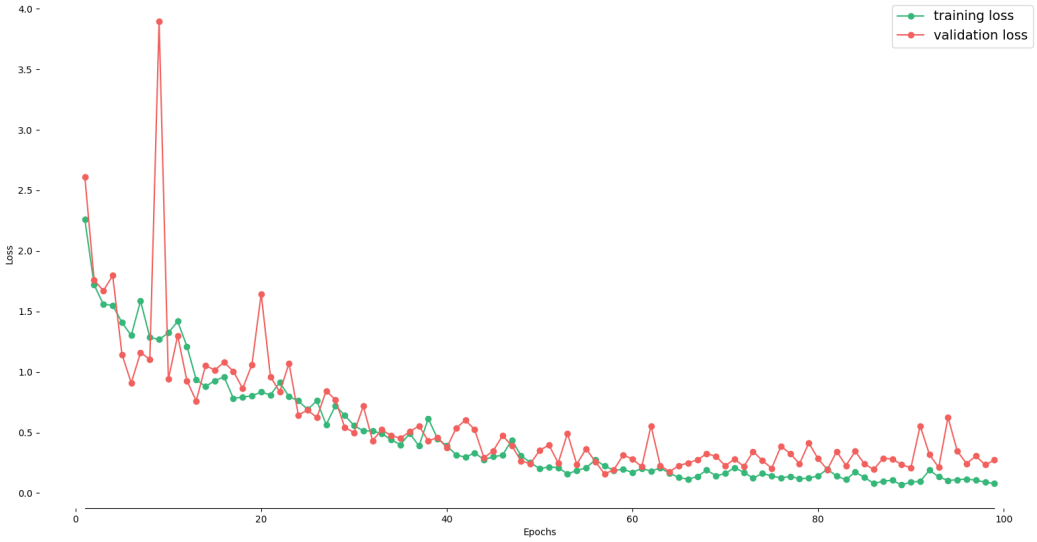


Figure B.5: Evolution of loss over time for Modelnet with 10 classes and 512 sampled points during fitting

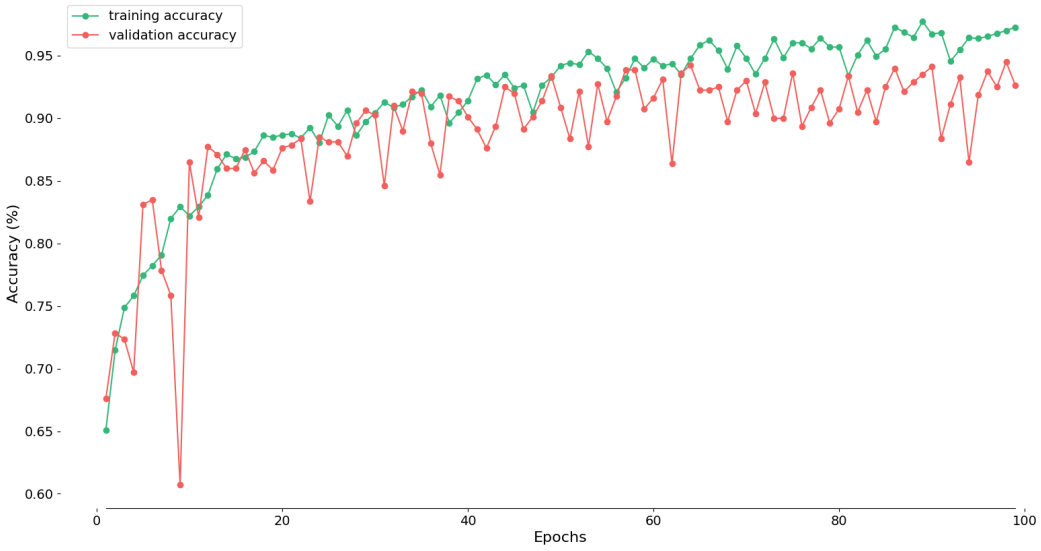


Figure B.6: Evolution of accuracy over time for Modelnet with 10 classes and 512 sampled points during fitting

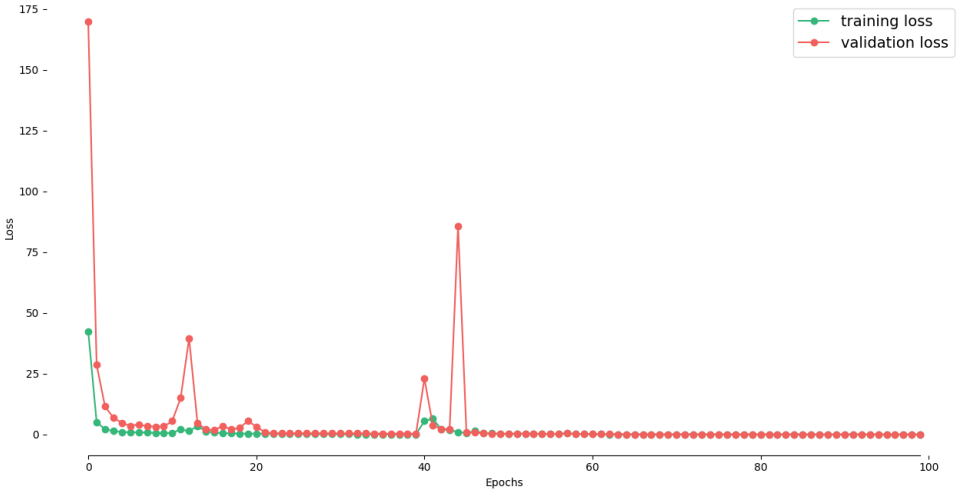


Figure B.7: Evolution of loss over time for Modelnet with 2 classes and 1024 sampled points during fitting

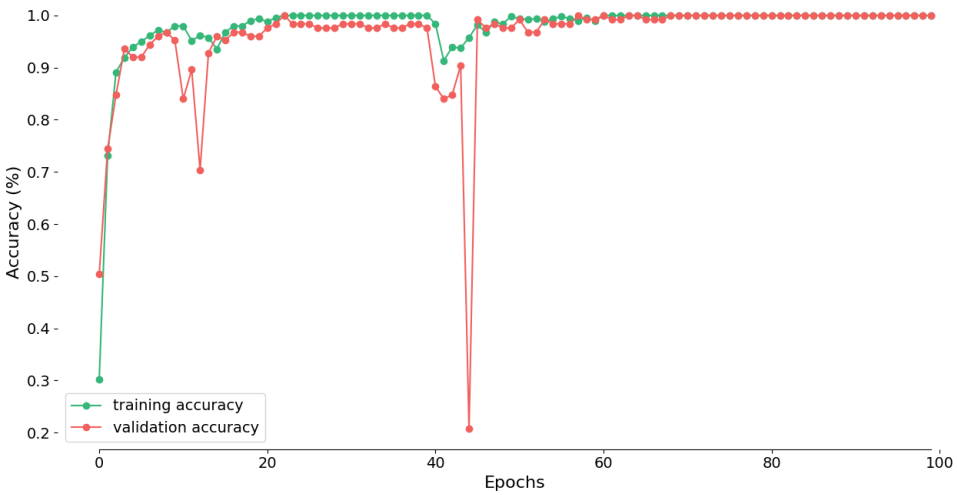


Figure B.8: Evolution of accuracy over time for Modelnet with 2 classes and 1024 sampled points during fitting

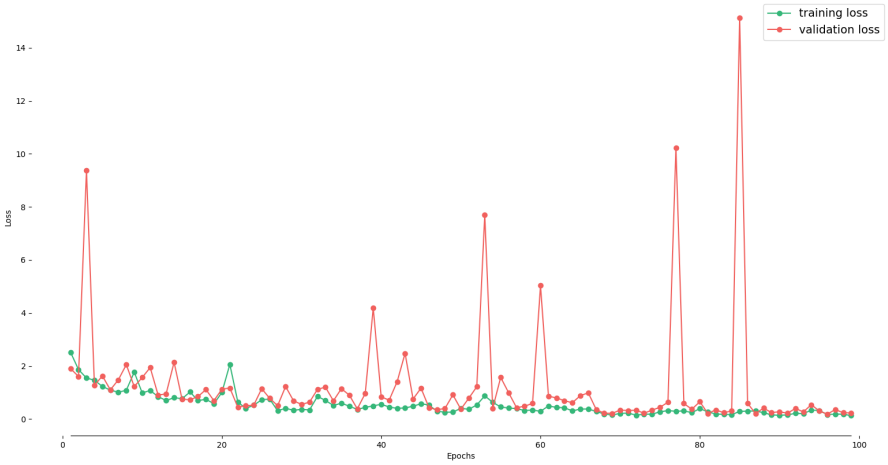


Figure B.9: Evolution of loss over time for Modelnet with 5 classes and 1024 sampled points during fitting

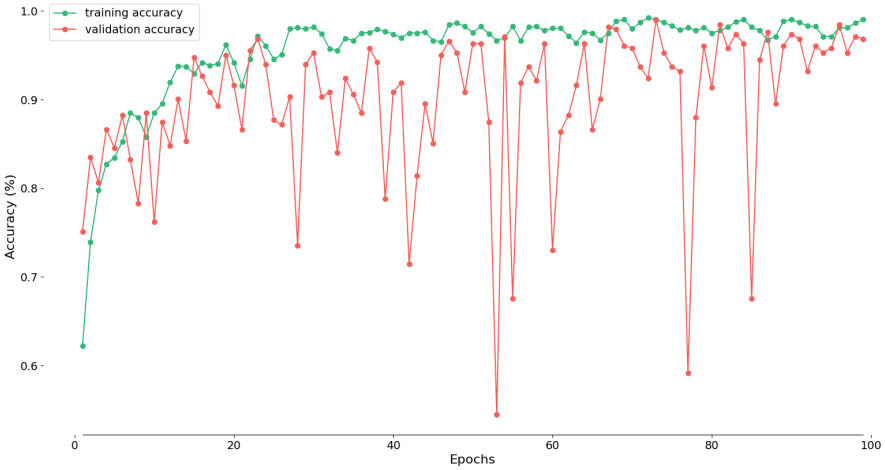


Figure B.10: Evolution of accuracy over time for Modelnet with 5 classes and 1024 sampled points during fitting

Bibliography

[1] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 652–660, 2017.

[2] J. Niemeyer, F. Rottensteiner, and U. Soergel. Conditional random fields for lidar point cloud classification in complex urban areas. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, I-3:263–268, 2012.

[3] D. Anguelov, B. Taskarf, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng. Discriminative learning of markov random fields for segmentation of 3d scan data. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 2, pages 169–176 vol. 2, 2005.

[4] Tony Lindeberg. Scale invariant feature transform. 2012.

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

[6] Samuele Salti, Federico Tombari, and Luigi Di Stefano. Shot: Unique signatures of histograms for surface and texture description. Computer Vision and Image Understanding, 125:251–264, 2014.

[7] M. Himmelsbach, T. Luettel, and H.-J. Wuensche. Real-time object classification in 3d point clouds using point feature histograms. In 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 994–1000, 2009.

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

[9] C Cortes. Support-vector networks. Machine Learning, 1995.

[10] Ruihu Wang. Adaboost for feature selection, classification and its relation with svm, a review. Physics Procedia, 25:800–807, 2012. International Conference on Solid State Devices and Materials Science, April 1-2, 2012, Macao.

[11] Zheng Gan, Liang Zhong, Yunfan Li, and Haiyan Guan. A random forest based method for urban object classification using lidar data and aerial imagery. In 2015 23rd International Conference on Geoinformatics, pages 1–4, 2015.

[12] Antonio Mucherino, Petraq J. Papajorgji, and Panos M. Pardalos. k-Nearest Neighbor Classification, pages 83–106. Springer New York, New York, NY, 2009.

[13] Marcel Körtgen, Gil-Joo Park, Marcin Novotni, and Reinhard Klein. 3d shape matching with 3d shape contexts. In The 7th central European seminar on computer graphics, volume 3, pages 5–17. Budmerice Slovakia, 2003.

[14] A.E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(5):433–449, 1999.

[15] Jean-François Lalonde, Nicolas Vandapel, Daniel F Huber, and Martial Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. Journal of field robotics, 23(10):839–861, 2006.

[16] Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. Margin based feature selection-theory and algorithms. In Proceedings of the twenty-first international conference on Machine learning, page 43, 2004.

[17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017.

[18] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[19] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. ACM Trans. Graph., 38(5), October 2019.

[20] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Learning semantic segmentation of large-scale point clouds with random sampling. IEEE Transactions on Pattern Analysis and Machine Intelligence, page 1–1, 2021.

[21] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-bert: Pre-training 3d point cloud transformers with masked point modeling. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 19313–19322, June 2022.

[22] Yatian Pang, Wenxiao Wang, Francis E. H. Tay, Wei Liu, Yonghong Tian, and Li Yuan. Masked autoencoders for point cloud self-supervised learning. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, Computer Vision – ECCV 2022, pages 604–621, Cham, 2022. Springer Nature Switzerland.

[23] Jiakai Fan, Weidong Song, Jinhe Zhang, Shangyu Sun, Guohui Jia, and Guang Jin. Pan: Improved pointnet++ for pavement crack information extraction. Electronics, 13(16), 2024.

[24] David Griffiths. pointnet2-tensorflow2. `https://github.com/dgriffiths3/pointnet2-tensorflow2`, 2020.

[25] Charles R. Qi. pointnet2. `https://github.com/charlesq34/pointnet2`, 2018.

[26] Ismail Khalid Kazmi, Lihua You, and Jian Jun Zhang. A survey of 2d and 3d shape descriptors. In 2013 10th International Conference Computer Graphics, Imaging and Visualization, pages 1–10, 2013.

[27] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent point feature histograms for 3d point clouds. In Intelligent Autonomous Systems 10, pages 119–128. IOS Press, 2008.

[28] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Learning informative point classes for the acquisition of object model maps. In 2008 10th International Conference on Control, Automation, Robotics and Vision, pages 643–650, 2008.

[29] Anastasia Ioannidou, Elisavet Chatzilari, Spiros Nikolopoulos, and Ioannis Kompatsiaris. Deep learning advances in computer vision with 3d data: A survey. ACM computing surveys (CSUR), 50(2):1–38, 2017.

[30] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43(12):4338–4364, 2021.

[31] Huang Zhang, Changshuo Wang, Shengwei Tian, Baoli Lu, Liping Zhang, Xin Ning, and Xiao Bai. Deep learning-based 3d point cloud classification: A systematic survey and outlook. Displays, 79:102456, 2023.

[32] Ying Li, Lingfei Ma, Zilong Zhong, Fei Liu, Michael A. Chapman, Dongpu Cao, and Jonathan Li. Deep learning for lidar point clouds in autonomous driving: A review. IEEE Transactions on Neural Networks and Learning Systems, 32(8):3412–3432, 2021.

[33] Junfei Wang, Luxin Hu, Xianfeng Wu, Zhongyuan Lai, and Qian Jia. Point cloud driven object classification: A review. In Shuo Yang and Huimin Lu, editors, Artificial Intelligence and Robotics, pages 260–270, Singapore, 2022. Springer Nature Singapore.

[34] LE Carvalho and Aldo von Wangenheim. 3d object recognition and classification: a systematic literature review. Pattern Analysis and Applications, 22:1243–1292, 2019.

[35] OpenAI. Chatgpt: A language model, 2024.

[36] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. arXiv preprint arXiv:1706.02413, 2017.

[37] Zhongbin Fang, Xiangtai Li, Xia Li, Joachim M Buhmann, Chen Change Loy, and Mengyuan Liu. Explore in-context learning for 3d point cloud understanding. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, Advances in Neural Information Processing Systems, volume 36, pages 42382–42395. Curran Associates, Inc., 2023.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**