Data structure-oriented learning strategies for 3D Objects classification

Andrea Lazzari*, Nicole Zattarin†, Nicola Zomer*

***Abstract*—Nowadays, the identification and understanding of 3D objects in real-world environments has a wide range of applications, including robotics and human-computer interaction. This is typically addressed using Deep Learning techniques that deal with 3D volumetric data, as they are generally able to outperform standard Machine Learning tools. In this work, we experiment with several architectures based on Convolutional Neural Networks with the aim of classifying 3D objects. We run our tests on the ModelNet40 [1] dataset, one of the most popular benchmark in the context of 3D object recognition. Furthermore, we take into account not only the model's performance but also its footprint, as the manipulation of 3D dataset has high-memory costs which require considering the number of parameters or GFLOPS among the relevant metrics. Going into detail, first we compare the effectiveness of Point Clouds and Voxel grids, inspecting pros and cons of these representations. We see how, for instance, the more accurate representation obtained via PC does not lead to better performance when dealing with CNNs, unless you have very large memory capacities. Then, we build an Autoencoder in order to retrieve an high-dimensional embedding of the input data. We show that the application of simple ML techniques, such as SVM, on these intermediate representations can lead to state-of-the-art performances and codewords could be used for compression purposes. Finally, we provide a visual representation of the encoded features through t-SNE.**

***Index Terms*—3D Object Classification, ModelNet40, Point Clouds, Voxels, Convolutional Neural Networks, Autoencoder**

I. Introduction

The problem of recognising objects in 3D is continuously increasing research attention due to its essential role in many real-world applications. Autonomous driving, intelligent robots and augmented/mixed reality are, indeed, just a few of the implementations based on this task. A simple reason for this is that 3D images are more meaningful and representative with respect to planar representations, since they describe concrete world scenarios carrying richer information. In addition, in the last few years the research community has been putting a lot of effort in developing new sensing technologies to collect 3D data, such as Stereo systems, Structured Light, Time of Flight sensors, and Light Detection and Ranging (LiDAR), thus supporting researchers to develop in this direction.

†Department of Information Engineering, University of Padova,
email: `nicole.zattarin@studenti.unipd.it`
*Department of Physics, University of Padova,
email: `andrea.lazzari.8@studenti.unipd.it`,
email: `nicola.zomer@studenti.unipd.it`

In the past years, Deep Learning techniques rapidly progressed in solving various tasks in computer vision and image processing, with lot of architectures based on Convolutional Neural Networks (CNNs). Models dealing with 3D volumetric data (e.g. the ones implemented in [2], [3], [4]) have been shown to achieve significantly better results compared to standard Machine Learning tools. Moreover CNNs have proved particularly well suited for this task, since they can explicitly exploit the spatial structure of 2D and 3D data, by learning local spatial filters useful in a classification framework.

Nowadays, in order to store 3D objects in a computerized ecosystem, different representations that vary both in structure and properties can be adopted (see Fig.1). In addition to Computer-Aided Design (CAD), which is useful for identifying the overall structure, one of the most widespread methods relies on *meshes*. Such method consists of a geometric procedure that allows the representation of surface subdivisions by a set of polygons, i.e. meshes. In detail, a mesh is made up of vertices (or vertex) connected by edges that form faces (or facets) of a polygonal shape. Despite their high cost in memory, meshes are notably used in computer graphics and modeling to represent surfaces.

Another way to represent 3D objects is *Point Clouds* (PC), which are nothing more then a set of data points in a three-dimensional coordinate system. These points are spatially defined by (*x, y, z*) coordinates and represent the main structure of an object. They are commonly obtained through Photogrammetry, LiDAR and more recently DL techniques through Generative Adversarial Networks. On the downside, although PC provide an exact representation of 3D objects, they are high memory consuming and sparse, as they tend to lack in connectivity.

In solid modeling, one of the most applied approach is, instead, *Voxelization*. A Voxel can be seen as a 3D base cubical unit that can be used to represent 3D models. Due to its 2D analogy with the pixel, a Voxel-based model can be seen as a discretized assembly of 3D pixels. What typically drives to adopt this representation is that PC makes it difficult to apply traditional Deep Learning framework. For each sample, traditional CNNs require its neighbors to appear at some fixed spatial orientations and distances so as to facilitate the convolution. Unfortunately, PC samples typically do not follow such constraints, being simply sparse. Then, one way to alleviate this problem is to Voxelize a PC to mimic the image representation and then to operate on Voxels. The downside is that Voxelization has to either sacrifice the representation accuracy or incur huge redundancies.

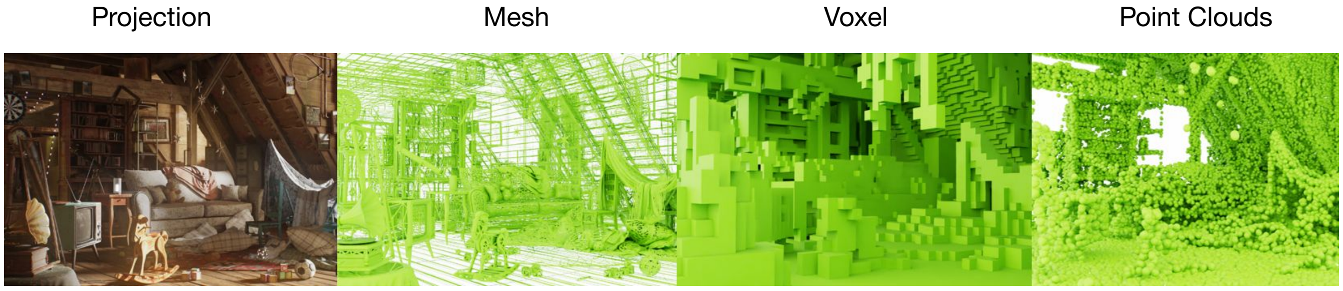


Fig. 1: Different representations of 3D objects (from the Kaolin library by NVIDIA). In particular, by means of projection we loose all the information concerning the depth and distances in the 3D environment. Meshes represent a good representation of the 3D structure via a polygon-like approximation of surfaces, while Voxels introduce a 3D pixelization of the environment. Finally, Point Clouds are a sparse, yet punctual, mapping of the 3D world.

In this framework, our work starts from the implementation of some CNN architectures that have been proven to be effective for the task of 3D object classification. In detail, motivated by [5], we implement a Voxel-based baseline model and we compare its performance with that of a PC-based CNN [3]. Inspired by the state of the art, we then move towards an Autoencoder (AE) model. We use the encoder to retrieve *codewords*, which can be seen as an high-dimensional embedding of the input data. This latter architecture can also be useful to discover similarities/common features across the various classes of the dataset, in order to inspect difficulties in the classification task and find pairs that are not easily distinguishable. We rely on t-SNE for this study [6].

Our approaches and implementations are tested on the ModelNet40 dataset [1], a typical benchmark for point cloud analysis. The remainder of this work is structured as follows. In Section II we describe the state of the art while a high level view of our strategy is presented in Sections III. Section IV is dedicated to data models while learning details and the used techniques are elucidated in Section V. The performance evaluation is carried out in Section VI while concluding remarks are provided in Section VII.

II. Related Work

The idea of applying CNNs to 3D object recognition is certainly not new. As a matter of fact, Ruizhongtai et al. [2] mentioned the importance of understanding 3D environments in computer vision scenarios and in many modern applications. Dealing with this problem, they studied the extension of 2D convolutional neural networks to 3D, showing how the additional computational complexity (volumetric domain) and data sparsity introduce significant challenges. For instance, in an image, every pixel contains observed information, whereas in 3D, a shape is only defined on its surface.

In a similar direction of our work, Sedaghat et al. [7] implemented 3D convolutional neural networks to classify objects in the ModelNet40 dataset. In detail, they analyzed both PC and voxelized images, focusing on inputs' orientation. In this context, they shown that rather shallow architectures and non-high resolution images can still provide results comparable with the state of the art. Moreover, to convert images to Voxels, they examined both binary-valued and continuous-valued occupancy grids, concluding that the differences between the two methods are negligible. Benefiting from this, we make the decision to use 0 or 1 binary Voxels.

Furthermore, a basic and effective 3D CNN architecture is presented in [5]. Inspired by them, we implement the *VoxNet* model, which requires the tranformation of input data from Point Cloud to Voxels. Regarding the size of the Voxel occupancy grid, we stick with their setting, characterized by a shape of $32 \times 32 \times 32$. This guarantees to avoid loss of shape information when the Voxel are too small (so that the object is larger than the grid) or when the Voxels are too large (so that details are lost by aliasing).

Such convolutional architectures require highly regular input data format, like those of image grids or 3D Voxels, to perform weight sharing and other kernel optimizations. As stated in [3], this conversion might introduce quantization artifacts which can mask data invariances, leading to network misinterpretation. This limitation is overcome by Qi et al. [3] by introducing the idea of using directly Point Clouds for classification. PC are just a set of points, each represented by three coordinates (x,y,z), which makes them robust to input permutations as well as to outliers or missing data. Moreover, they keep the representation accurate and they do not require heavy redundancy, as Voxels do. For these reasons, we believe it is interesting to compare PointNet and VoxelNet architectures to understand if the use of PC actually leads to a benefit in terms of both performance and computational cost.

Finally, in our last approach we take inspiration from Yang et al. [8], who focused their research on the emerging field of unsupervised learning for PC. In their study, they proposed an auto-encoder (AE) architecture, *FoldingNet*, that can be used to obtain an high-dimensional embedding of the input data. This latent representations can then be feed to a simple classifier (e.g SVM or MLP) in order to tackle the classification task. More in detail, they showed that FoldingNet outperforms all other methods on ModelNet40, proving its efficiency in representation learning and feature extraction from 3D PC.

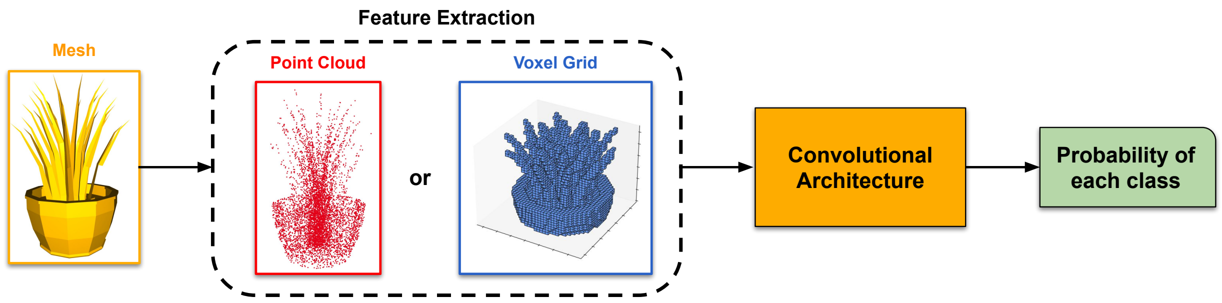


Fig. 2: High-level overview of the processing pipeline. The mesh is first converted into either a Point Cloud or a Voxel grid, then data go thorough a specific architecture, which are mainly Convolutional. The final label is obtained with either a purely supervised approach or an unsupervised DL strategy combined with an easier supervised model (e.g. SVM).

III. Processing Pipeline

It is well known that Neural Networks can address the classification task both in a supervised and in an unsupervised fashion. In the first case, the output of a Network are N logits, being N the number of classes considered. Logits are then used to compute a probability distribution over the set of classes by applying Softmax. Moreover, in the unsupervised scenario, it is possible to train an Autoencoder on the reconstruction task and retrieve the codewords to perform the actual classification. Indeed, in this second case, codewords represent a set of features that can be employed in any ML and NN-based classifiers, such as Support Vector Machines (SVMs) and Multi Layer Perceptron (MLP). For each of these approaches, the supervised and the unsupervised one, we provide information on two levels: with Voxel-based and with Point Cloud-based architectures. The former approach is addressed on a 2D level, while the latter with 3D architectures. An high level overview of the processing pipeline that brings from the mesh to the prediction is provided in Figure 2.

IV. Signals and Features

We employ the well-known ModelNet40 dataset [1], a comprehensive and clean collection of 3D CAD models of 40 popular object categories. It consists of 12311 distinct CAD models belonging to 40 classes, 9843 elements are for training and 2468 models for testing. In Point Cloud applications, for computational resources reasons, we select 4000 PC for training and 2468 PC for testing. As for Voxels, we build a downsampled version of ModelNet40, with 5878 elements for training and 1295 for testing.

To give a visual idea of the kind of objects we have to deal with, Figure 3 shows examples of meshes realized with Open3D [9]. On of the main difficulties in approaching this class of data is related to the choice of the 3D-object representation, i.e. the trasformation from meshes to Voxels or PC. Moreover, let us highlight that the dataset itself contains some classes which are characterized by common features, e.g. *flower pot* and *plant*, or *table* and *desk*.

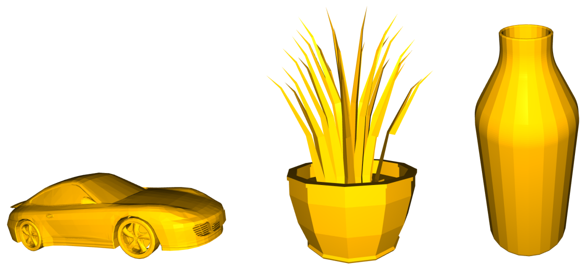


Fig. 3: Visualization of meshes [9] of a car, a plant and a bottle of the ModelNet40 dataset [1].

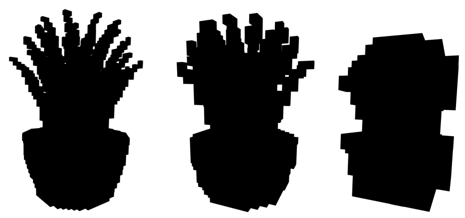


Fig. 4: Visualization of a mesh from ModelNet40 with voxel size 1, 2, 5 (from left to right). Clearly, increasing the voxel size reduces the resolution of the mesh. Moreover, a voxel size equal to 1 is a good trade off in terms of quality and computational resources needed.

A. Voxelization

The main preprocessing for Voxel-based applications consists in the conversion from meshes to Voxels. This process is usually referred as *voxelization*. To this end, we apply the strategy proposed in [10], [11]: generated Voxels have shape $32 \times 32 \times 32$ and the Voxel size is fixed to 1, since it turns out to be a good trade-off between computational complexity and resolution. To support such choice, in Figure 4, we show the same object with different Voxel sizes.

B. Point Clouds

In ModelNet40 each file provides all the data needed to reconstruct the meshes. Therefore, the most straightforward approach to extract PCs consists of keeping the vertices of meshes as elements of the PC. However, since PCs are computationally heavy to process, to speed up the learning and to adjust to our memory resources, for each PC we consider a subsample of 4000 points. Each downsampled PC is obtained by uniformly sampling points from the original PC.

V. Learning Framework

As we already pointed out, for both 3D-object representations, we explore both a supervised and unsupervised framework. In the this section, we describe the architectures designed to address the task in the different scenarios.

A. Supervised learning scenario

The most straightforward approach to classification consists of training a network whose outcome is the probability distribution over the set of classes. The network outputs are series of N logits, where N the number of classes in the dataset. Logits are then input to a *Softmax* layer, which returns the probability that the original sample belongs to each class k.

In particular, given an input element $\mathbf{x}_n$, define $t_{n,k} = 1$ if $\mathbf{x}_n$ belongs to class k, else $t_{n,k} = 0$. The probability that $\mathbf{x}_n$ belongs to k is given by:

$$P(t_{n,k} = 1|\mathbf{x}_n) = y_{n,k} = \frac{e^{a_L^k}}{\sum_{j=0}^{N} e^{a_L^j}}, \tag{1}$$

where a_L^k is the k-th output of layer L, i.e. output of the last *Dense* layer. Once the probability distribution is known, the model predicts that the input $\mathbf{x}_n$ belongs to class q, where q is obtained as follows:

$$q = \arg\max_k \left[P(t_{n,k} = 1|\mathbf{x}_n)\right] = \arg\max_k \left(y_{n,k}\right). \tag{2}$$

In simple terms: for each sample the model output the probability distribution among classes and the label corresponding to the highest probability.

1) Voxel-based model: Regarding Voxel-based architectures, we provide two different strategies, both inspired from VoxNet [5]. The full architecture is shown in Figure 5: the Voxel goes through a series of 3D Convolutional layers, together with MaxPooling operation and a final linear section. Convolutions are followed by dropouts with probabilities $[0.5, 0.6, 0.7]$, LeakyRelu is used as activation function. The linear section is designed in order to narrow down the tensor dimension to N, being N the number of classes in the dataset, i.e. to generate logits needed for classification.

As a second strategy, we train and test a similar architecture enriched by residual connections. The main Network architecture follows the one described in Figure 5, where, excluding the first one, after all the others Convolutional blocks C_k a residual connection is added. If we call respectively O_{C_k} and I_{C_k} the output/input of C_k, then the input of the next Convolutional block is given by:

$$I_{C_{k+1}} = ReLU\left(O_{C_k} + I_{C_k}\right). \tag{3}$$

Channel shapes and kernel sizes are adjusted in order to fit the proper tensor shapes without introducing paddings.

In both cases, training is performed with a Cross Entropy, while Adam is used for optimization, details concerning parameters are shown in Table 1.

2) Point Cloud-based model: Common approaches for PC classification are inspired by PointNet [3]. We follow a similar path and implement the architecture shown in Figure 6. First, the PC is downsampled to 4000 points, in order to fit our computational resources and allow for a faster training. Each sample is processed as a 2D element, where the first dimension is given by the number of points and the second dimension stands for the coordinate, i.e. the triplet (x, y, z). The tensor goes through multiple Convolutions, followed by a MaxPooling layer and a final linear section, needed to generate the logits. As a regularization strategy, each convolution is followed by a Batch Normalization layer, and a ReLU activation function is then applied. Training is performed with Stochastic Gradient Descent (SGD), as after different tests it turns out to be the less overfitting optimization strategy, even tough it is less stable. Parameters details are again shown in Table 1.

B. Unsupervised learning scenario

Neural Networks are, at the end of the day, features extractors. Therefore, they represent a powerful tool to learn features vectors which can be employed in any DL- and ML-based classifier.

1) Autoencoder reconstruction: We propose to train Autoencoders (AEs) on the reconstruction task and to feed the learned codewords into SVM and MLP models to perform classification. Let us assume that the encoding corresponds to a function $\phi(\cdot)$, while the decoding to a function $\psi(\cdot)$. Therefore, given a input tensor $\mathbf{x}_n$, the AE performs the following operations:

$$\begin{aligned} \mathbf{x}_n &\rightarrow \mathbf{c} = \phi(\mathbf{x}_n, \mathbf{W}^{\phi}) \\ &\rightarrow \tilde{\mathbf{x}}_n = \psi(\phi(\mathbf{x}_n, \mathbf{W}^{\phi}), \mathbf{W}^{\psi}) = \psi(\mathbf{c}, \mathbf{W}^{\psi}), \end{aligned} \tag{4}$$

where $\mathbf{W}^{\phi}$ and $\mathbf{W}^{\psi}$ are respectively encoder and decoder weights, while $\mathbf{c}$ is the codeword. If distance in the original space is measured with a distance function D, then the loss function to optimize is the distance between the input and the output, in formula:

$$\mathcal{L}\left(\mathbf{x}_n, \mathbf{W}^{\phi}, \mathbf{W}^{\psi}\right) = D\left(\mathbf{x}_n, \tilde{\mathbf{x}}_n\right). \tag{5}$$

In our application, we train the AE in order to minimize such loss function. Then, we keep the codewords $\mathbf{c}$, feeding them to the actual classifiers.

a) Voxel-based AE: The Voxel-based AE architecture is made up by three encoding Convolutional blocks, each of which consists of a 3D convolution, a Batch Normalization layer and Dropout. The codeword is learned with a final linear section, which narrows down the dimension to 512. The decoder has the exact same structure, bottom up. Since Voxels are 3D binary matrices, in this case we simply make use of MSE as distance metric. Training is performed with Adam, learning rate is 10^{-3}, and weight decay is 10^{-6}.

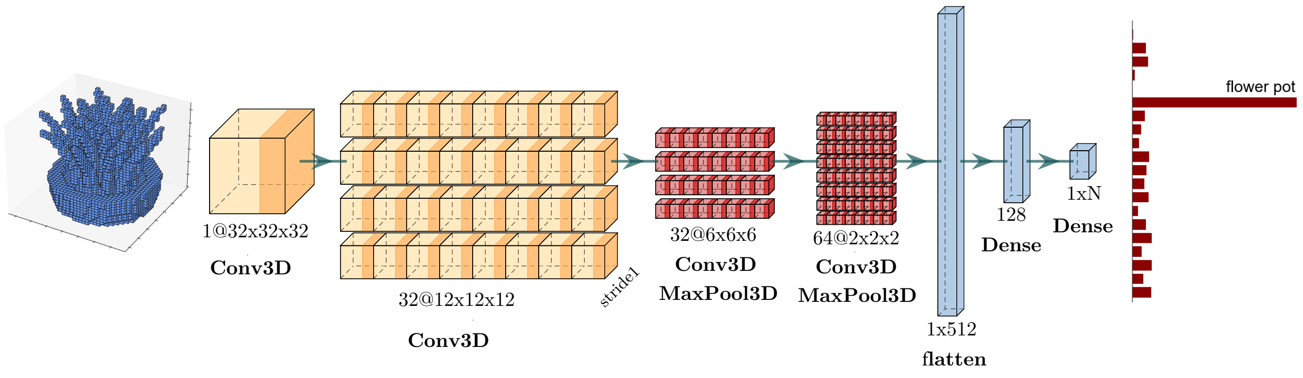


Fig. 5: CNN based architecture employed to process and classify Voxels. The 3D Voxel first goes through 3D Convolutions, followed by pooling, then the tensor is narrowed down to N, being N the number of classes. In order to reduce overfitting, Convolutions are followed by Dropout, with probabilities $[0.5, 0.6, 0.7]$. LeakyRelu is used as activation function.

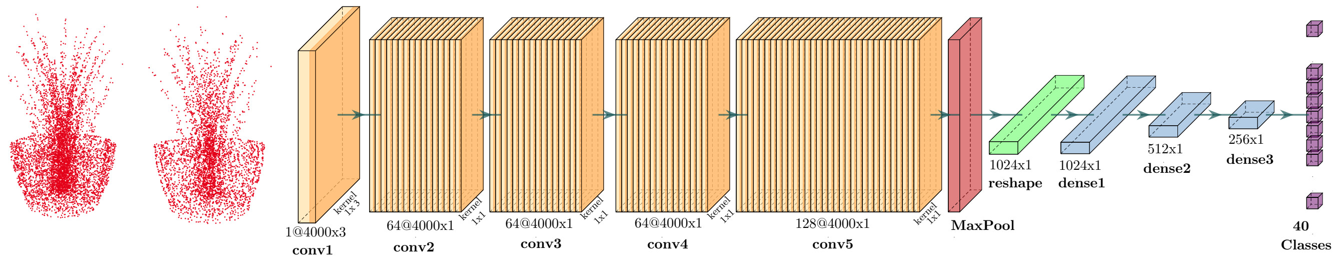


Fig. 6: CNN based architecture employed to process and classify Point Clouds. The PC is first downsampled to 4000 points, then 2D Convolutions are applied, followed by a MaxPooling layer. Finally, the tensor is narrowed down to N, being N the number of classes.

b) Point Cloud-based AE: The Point Cloud-based architecture is inspired by [8]. Following the approach we presented in Equation 5, the quality of the reconstruction is measured by means of Chamfer distance [12], which is a common choice in PC scenarios. Nevertheless, from a computational point of view, Chamfer distance is extremely slow to compute and it generally needs GPU acceleration. In particular, considering an input PC, namely $\mathcal{S}$, and its reconstruction $\tilde{\mathcal{S}}$, Chamfer distance D_{CH} is given by:

$$D_{CH}(\mathcal{S}, \tilde{\mathcal{S}}) = \max \left[\frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \min_{\tilde{x} \in \tilde{\mathcal{S}}} \|x - \tilde{x}\|_2, \right. \left. \frac{1}{|\tilde{\mathcal{S}}|} \sum_{\tilde{x} \in \tilde{\mathcal{S}}} \min_{x \in \mathcal{S}} \|\tilde{x} - x\|_2 \right]. \tag{6}$$

The architecture consists of a series of Convolutional blocks with unitary kernels, in which each Convolution is followed by a Batch Normalization layer for regularization purposes. In detail, the encoder's input goes through three Convolutional blocks and a MaxPooling layer. Then, the output of the first block is concatenated with the MaxPooling output and feed into two fully connected layers. In such way, codewords of size 512 are created. On the other side, the decoder has the exact same structure, bottom up. Training is performed for 128 epochs with Adam optimizer. Learning rate is set to 10^{-3} and weight decay to 10^{-6}.

2) Classification: Once the AEs are trained for the reconstruction task, we perform classification using a simple classifier, taking in input the codewords as feature vectors. As far as this classifier is concerned, our default choice are SVMs. SVM parameters are obtained with a grid search and then fixed. Finally, they are tested on the entire test dataset.

For what concerns the PC-based architecture, SVM is tested using a polynomial kernel, regularization parameter $C = 1$ and kernel coefficient $\gamma = 0.01$. The Voxel-based SVM is, instead, tested selecting a linear kernel, setting a regularization parameter $C = 10$ and a kernel coefficient $\gamma = 0.001$.

As an alternative, only with respect to the PC-based architecture, we train a small FF classifier. Its architecture consists of four Dense layers, combined with Batch Normalization and Dropout. In this framework, ReLu activation function and Adam optimizer are used.

As a last step, we perform a dimensionality reduction on the retrieved codewords. This is implemented in order to visualize the quality of the clustering that can be achieved with the learned features. To this end, we apply t-SNE [6], with 2000 iterations and perplexity value equal to 20.

Architecture	optimizer	Epochs	lr	weight decay
PC-based	SGD	47	0.001	0.00001
Voxel-based	Adam	330	0.0001	0.00001
Res-Voxel-based	Adam	80	0.0001	0.00001

TABLE 1: Training details for each of the architectures taken into account for the supervised learning approach. Adam is usually the preferred choice, nevertheless we choose to train the PC-architecture with SGD as it turns out to better converge in our case.

Architecture	Test Accuracy	Precision	Recall	F1
PC-based	0.82	0.71	0.72	0.71
Voxel-based	0.84	0.73	0.74	0.72
Res-Voxel-based	0.83	0.72	0.75	0.71

TABLE 2: Results for each of the architectures taken into account for the supervised learning approach. All three models considered show similar performances in general, with slightly different values for the computed metrics.

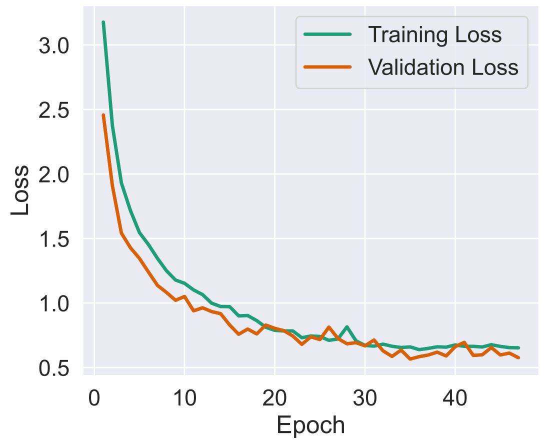
VI. Results

Before going through a detailed discussion of results, let us anticipate that the methods presented here show similar performances, even tough they exploit different features. Moreover, let us recall that our models are slightly less performing than the state-of-art, mostly because we considered fewer amount of data and we downsampled PCs for computational reasons. Despite this, the results are still significant. Finally, the models' footprint is also included in the discussion.

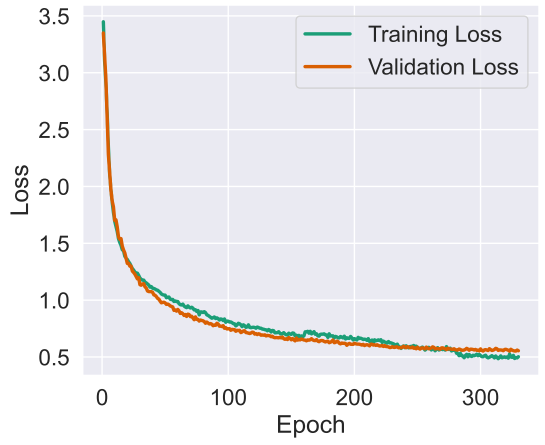
A. Supervised learning scenario

As for the supervised learning approach, Figure 7 shows how the loss functions (i.e. cross entropy) evolve over epochs for Point Cloud-based and Voxel-based classifiers. Let us highlight that the learning curves are smoother in the second case, in which Adam has been applied, while SGD shows a less stable optimization. Nevertheless, both models exhibit a similar trend and achieve similar loss function values. In particular, we trained the models in such a way to reach a good train-validation trade-off and avoid overfitting. This is also why we choose SGD instead of Adam when training our PC-based model. Note, recalling Table 1, that training has a different number of epochs in the three cases. Indeed, each model has its own specific learning curves and needs care in the training process, therefore we pick a proper number of epochs according to each specific case.

In Table 2 we report the metrics obtained on the entire test dataset for the three methods considered. Observe how all results are comparable in terms of performances. Even the inclusion of skip-connections does not lead to an improvement in terms of accuracy in this case. In particular, recall that, since we are dealing with 40 classes, the accuracy in a random scenario would be 0.025. Therefore, an accuracy of 84% corresponds to about $82\% - 81\%$ improvement over the "flip coin" case.



(a) Point Cloud-based architecture training curves. Optimization is performed with SGD, as it turns out to be the best strategy to avoid overfitting, at the cost of loosing stability in the learning process.



(b) Voxel-based architecture training curves. The learning is smooth and reaches a flat trend after the considered number of epoches.

Fig. 7: Evolution of validation and training loss functions through epochs for PC- and Voxel-based architectures.

B. Unsupervised learning scenario

For what concerns the unsupervised learning approach, we observe similar performances in classification to those highlighted in Section VI-A. Detailed metrics are shown in Table 3. The main outcome is that, in this case, PC-based strategies are much better performing than Voxel-based ones.

To visualize the quality of the features obtained in the AE training, we perform a 2-components dimensionality reduction with t-SNE [6]. In particular, in order to have a clear visualization, we select a subsample of classes of ModelNet40. In Figure 8, we see that, in the PC-based case, codewords show good performances in capturing features in the original PC. Indeed, certain labels are clearly clusterized in a single cloud, e.g. `airplane` and `chair`, while some others create multiple small clusters for the same label, see `guitar`. Moreover, note that separation is drastically lower in the case of similar objects, e.g. `desk` and `table`, since codewords tend to individuate the same features. We could expect such results, considering that some of these objects are difficult to distinguish even for humans, as it is pointed out in [8]. On the other side, the Voxel-based approach turns out to be way less performing. In fact, the t-SNE visualization shows that certain labels are clearly clustered, e.g. `airplane`, but in general we observe a lot of overlaps between different clouds. Note also that in some cases, e.g. `laptop`, t-SNE actually individuate a cluster, but it is not well distinguishable from others labels.

The difference in performances between Voxel- and PC-based could be explainable in terms of data structures: even tough PC are sparse and their processing is more computationally demanding, Voxels introduce a simplification, i.e. the Voxel grid size, which does not guarantee a good reconstruction. In particular, we observe that high dimensional objects, such as `car` and `airplane`, are better clustered than other classes. This is still explainable considering that the Voxel size approximation in these cases causes a lower loss in terms of resolution.

C. Computational resources observations

Finally, let us now focus on the computational resources employed. In Table 4 we show the total number of parameters of each architecture and the corresponding Giga FLoating point Operations Per Second (GFLOPS), i.e. the metric commonly used to estimate the speed of a computer in units of billion floating-point operations per second. As for Autoencoders, we report only the computational cost of the encoder, since it is the part of the architecture used for the classification task.

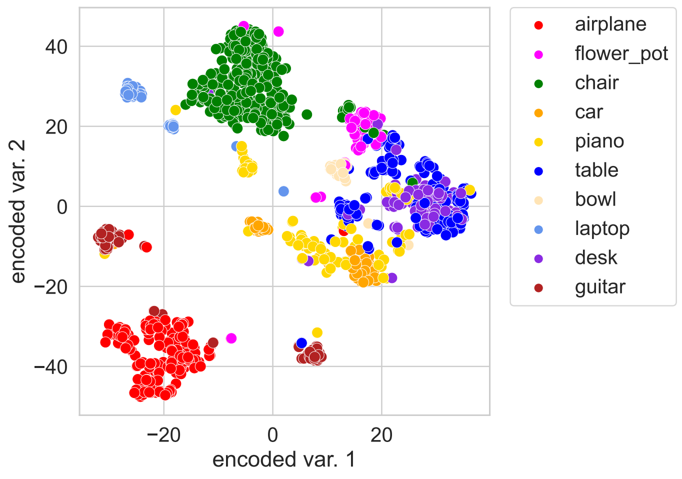
Moreover, again from Table 4, we can notice that the PC-based designs are more computationally expensive (both in terms of parameters and GFLOPS) with respect to the ones dealing with Voxels. Indeed, PC-based models have to account for the intrinsic sparsity of PCs, which makes it difficult to individuate connections in samples. In addition, the data itself has larger shape in the PC format, since each sample contains 4000 points, each of which has 3 dimensions, while Voxels are narrowed down to a 32x32x32 grid.

A more in-depth discussion of computational resources metrics in relation to performance is presented in Section VII.

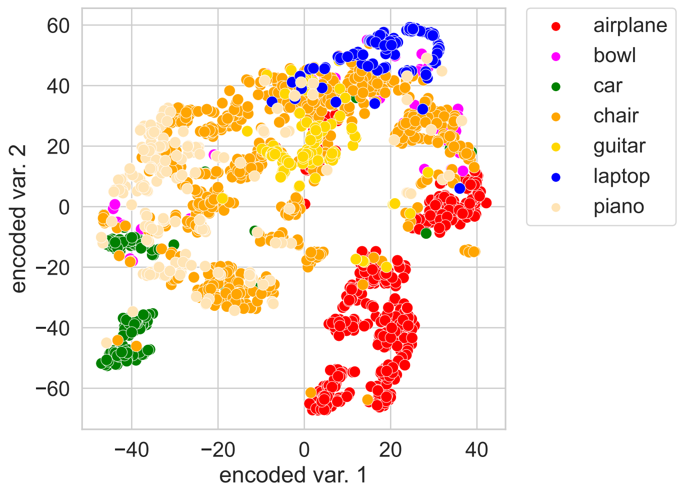
VII. Concluding Remarks

In this project we tackled the 3D object recognition task focusing on different 3D representations and comparing multiple CNN-based architectures. We started by implementing two Voxel-based model: a CNN and a ResNet both based on VoxNet [5]. We have seen how the introductions of skip-connections is not justified by the performance results. Indeed, we got a 1% improvement in the test accuracy with a relative increase of more then 15% in the number of parameters. We then moved towards a PC-based CNN architecture, based on PointNet [3], with the aim of testing if the use of a more accurate representation helps the model to learn. However, the test accuracy dropped by 2% with an increase of more than 4× in the number parameters. Despite that, we believe it is worth it to continue research in this direction, as it is known that PC-based models can outperform VoxNet when more points are considered to design the PC. In this scenario, an interesting line of research would be to investigate the impact of different sampling strategies in the creation of the PC, keeping the number of points fixed and small for memory reasons.

As a last approach, we considered the implementation of an Autoencoder model, based on [8]. First, in support of the previous conclusions, we showed that in this case the use of PC is strongly motivated by the performance. In detail, PC



(a) Point Cloud-based AE codewords, t-SNE visualization



(b) Voxel-based AE codewords, t-SNE visualization

Fig. 8: Dimensionality reduction obtained with t-SNE on a selected subsample of codewords corresponding to 10 or less classes of ModelNet40. Color is given by the class label. It is clearly possible to see that codewords are a good representation of the original PC in a lower dimensional space. By applying t-SNE on the PC-based codewords we notice that classes are clearly distinguishable, and in general points of the same class tend to be clustered together.

Architecture	Test Accuracy	Precision	Recall	F1
PC-based AE + SVM	0.82	0.74	0.76	0.74
PC-based AE + FFNN	0.79	0.71	0.74	0.71
Voxel-based AE + SVM	0.67	0.53	0.58	0.53

TABLE 3: Results for each of the AE plus simple classifier architectures. For the unsupervised learning approach, we take into account a Voxel-based and a PC-base AE, while for classification SVM and a small FFNN are considered.

Architecture	Total Parameters	GFLOPS
PC-based	818 K	304
Voxel-based	158 K	11
Res-Voxel-based	2.8 M	248
PC-based AE	1 M	7652
Voxel-based AE	11 M	386

TABLE 4: Total number of parameters and GFLOPS for each of the designed architectures. In the case of the AE, we take into account only the parameters related to the Encoder part of the network.

allows to obtain a relative increase in the accuracy of more than 20% with respect to a Voxel-based AE, using ≈10x less parameters. We believe this is an effect of the poor resolution of Voxels. Indeed, the loss of spatial information deriving from voxelization probably complicates the AE task of obtaining an even more compressed embedded representation of the input objects. Finally, AE models are shown to be effective in the compression of input 3D objects (see Figure 8). This could be useful not only for extracting features to apply the classification task on, but also for obtaining a compressed representation of the objects, required for example by low-memory devices.

A. *Applications and future development*

3D object classification represents an interesting research topic for Autonomous Driving applications, since the techniques and models needed to address such task are similar to those needed in Semantic Segmentation (SS) and Vehicle-to-Everything (V2X) communication. Indeed, all these models have to deal with 3D object representation and with the developing of Deep Learning architectures which allow to retrieve a proper understanding of the surrounding environment.

In particular, connected vehicles will be equipped with different kind of sensors, e.g. Light Detection and Ranging (LiDAR) and RGB camera, that are employed to map the environment around the vehicle itself. In particular, LiDARS are a prominent technology, widely employed to generate the PC corresponding to the surrounding world. Such PC then need to be efficiently shared among communication channels, processed, and feed into specific NN models for SS and more. In this pipeline, classification itself represents a minor element in such pipeline, but it is worth to show the big picture behind the work developed in this paper.

Therefore, a possible further application of our work could deal with training our models on Autonomous Driving datasets, such as SELMA [13].

B. *Learning Process and Adversities*

In this analysis we took confidence with different 3D representations inspecting the advantages and the drawbacks in a classification framework. In particular we made a comparison between several approaches using Point Clouds and Voxels in order to study various CNN-based architectures. Management of these kinds of data was a useful competence to add. The major difficulties we have encountered were related to the high computational time and memory costs of the 3D dataset we analyzed.

Individual contribution

We worked together as a team, from brainstorming and reviewing the literature to developing a proper strategy in order to approach the task. Moreover, each of us brought its personal contribution according to each individual experience and interests. In particular, Nicole already had experience in 3D Augmented Reality strategies and Neural Network architectures for Point Cloud processing. While Andrea reviews all the various methods of 3D representation; he also deals with the training of the various models and taking care of the graphical part of the report. Finally Nicola designed and tested multiple learning strategies in order to find the best optimization approaches and developed high quality conclusive remarks.

The paper has been written all together.

References

[1] Z. Wu, S. Song, A. Khosla, X. Tang, and J. Xiao, "3d shapenets for 2.5d object recognition and next-best-view prediction," *CoRR*, vol. abs/1406.5670, 2014. [Online]. Available: http://arxiv.org/abs/1406.5670

[2] C. Ruizhongtai Qi, H. Su, M. NieBner, A. Dai, M. Yan, and L. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," 06 2016, pp. 5648–5656. [Online]. Available: https://arxiv.org/abs/1604.03265

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

[4] V. Hegde and R. Zadeh, “Fusionnet: 3d object classification using multiple data representations,” *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1607.05695

[5] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 922–928.

[6] L. van der Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html

[7] N. Sedaghat, M. Zolfaghari, and T. Brox, “Orientation-boosted voxel nets for 3d object recognition,” *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1604.03351

[8] Y. Yang, C. Feng, Y. Shen, and D. Tian, “Foldingnet: Interpretable unsupervised learning on 3d point clouds,” *CoRR*, vol. abs/1712.07262, 2017. [Online]. Available: http://arxiv.org/abs/1712.07262

[9] Q. Zhou, J. Park, and V. Koltun, “Open3d: A modern library for 3d data processing,” *CoRR*, vol. abs/1801.09847, 2018. [Online]. Available: http://arxiv.org/abs/1801.09847

[10] P. Min, “binvox,” `http://www.patrickmin.com/binvox` or `https://www.google.com/search?q=binvox`, 2004 - 2019.

[11] F. S. Nooruddin and G. Turk, “Simplification and repair of polygonal models using volumetric techniques,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 191–205, 2003.

[12] A. Hajdu, L. Hajdu, and R. Tijdeman, “Approximations of the euclidean distance by chamfer distances,” 2012. [Online]. Available: https://arxiv.org/abs/1201.0876

[13] P. Testolina, F. Barbato, U. Michieli, M. Giordani, P. Zanuttigh, and M. Zorzi, “Selma: Semantic large-scale multimodal acquisitions in variable weather, daytime and viewpoints,” 2022. [Online]. Available: https://arxiv.org/abs/2204.09788

[14] X. He, Y. Zhou, Z. Zhou, S. Bai, and X. Bai, “Triplet-center loss for multi-view 3d object retrieval,” *CoRR*, vol. abs/1803.06189, 2018. [Online]. Available: http://arxiv.org/abs/1803.06189