Voxel based 3D object classification: a focus on orientation boosting and fast hybrid cascade network

Andrea Peri[†], Marija Mojsovska[†]

***Abstract*—The 3D object classification task is gaining increasing interest nowadays, with its application ranging from video and 3D scene segmentation to automatic robot navigation in real world scenarios. In this report, we present a convolutional approach to 3D object classification based on the VoxNet architecture. Then, we take a step further towards the study on the relevance of object orientation to improve classification performances, resulting in the implementation of the Orion architecture. We show that a convolutional neural network is able to achieve good results on the ModelNet10 and ModelNet40 datasets. On the other hand, we find that the magnitude of the improvements introduced by Orion is below our expectations. The convolutional approach is effective but it is quite computational and time demanding. We present a solution to tackle this problem, introducing a hybrid cascade network composed of a multilayer perceptron and the Orion architecture. We show that this approach is effective in reaching lower inference time while maintaning a high accuracy.**

***Index Terms*—3D Object Classification, Convolutional Neural Networks, Binary Voxel Grid, Hybrid Cascade Architecture .**

I. Introduction

Machine learning-based 3D object classification has gained increasing interest in recent years, both in the academia and in the industrial sector. This is mainly due to advancements in the deep learning field. Indeed, a deep learning architecture allows to automatically extract relevant features from observed data, enabling the extraction of much richer attributes and releaving researchers and developers from the necessity to design specific filters for different tasks.

In real-world applications, 3D data are usually acquired with depth cameras or lidar sensors. Thus, contrary to images, properties as colour, texture, etc. are usually meaningless in the 3D classification task. It appears clear that the most important feature is the shape of the object. Another important attribute of a 3D object is its viewpoint. Object category and its orientation are extremely connected features, and an algorithm trained to perform category classification should properly take into account object rotations.

Altough the 3D object classification task can be solved in different ways, an intuitive and effective approach to tackle the problem is to extend the ideas of 2D image classification. From such field we inherit two basic primitives on which our learning framework is based:

- **Voxel-based representation of the dataset**: voxels are the 3D counterpart of pixels in an image;
- **Convolutional layers**: this type of layers has shown to be extremely effective in the 2D classification task, since they are able to exploit the structural information contained in multi-dimensional objects.

Although voxel-grids are a 3D data format able to easily store structural information, their usage present some challenges. Indeed, a convolutional approach to voxel-grids is a computationally demanding task. For this reason any strategy to decrease the inference time is appreciated. With inference time we refer to the amount of time it takes for a trained model to make a prediction.

From these starting points we developed our project as follows:

- **implementation of VoxNet** [1], a shallow convolutional network trained to predict object category;
- **implementation of Orion** [2], an architecture based on VoxNet and trained to predict the object viewpoint as well;
- **implementation of a hybrid cascade architecture**: this is an attempt to reduce inference time in object classification, inspired by [3]. This approach consists in the concatenation of a shallow fully connected network with the Orion architecture.

Our work can be described as follows. First, we implement VoxNet and we train it on ModelNet10 and ModelNet40 datasets. The second step is to train the Orion architecture on the same datasets to evaluate the effectiveness of the viewpoint-oriented approach described in [2]. Lastly we implement the cascade architecture to show that a significant improvement in inference time can be achieved, while maintaining a high accuracy.

This report is organized as follows. In Sec. II we will review the literature on the 3D object classification field, with focus on the voxel-based approach. We will provide a description of our learning framework in Sec. III, from the network architectures to specific ideas we implemented. In Sec. IV we describe the datasets and the pre-processing operated on them and in Sec. V we describe the hyper-parameter optimization process. Finally, we will provide the results of our project in Sec. VI. In Sec. VII and VIII we present our concluding remarks and observations.

II. Related Work

The 3D object classification task can be approached with different methods, mainly based on the object representation format. In the literature, a widely explored solution is to render

[†]Physics of Data, ID: 2063592,
email: andrea.peri.1@studenti.unipd.it
[†]Physics of Data, ID: 2044720,
email: marija.mojsovska@studenti.unipd.it

3D samples as point clouds, namely a collection of points in a 3D space ([4], [5]). Point clouds are a promising format for 3D representation especially in the machine learning field, due to their lightweight nature in terms of memory usage and computational requirements. On the other side, point clouds presents some problems in the representation of 3D spatial structures.

A different approach to 3D object classification is the representation of objects as voxel grids, which are a natural extension of 2D pixel images in a 3D space. Furthermore, this data structure can be easily processed by a 3D convolutional network. One of the first important contribution in the voxel-based 3D classification problem is given by [1]. The authors presented a shallow convolutional network, composed of two convolutional layers followed by two fully connected layers. With this simple structure, they are able to obtain state of the art results in various 3D datasets. As it was already clear at the time the paper was published, multi-views are an extremely important concept when dealing with 3D objects. The authors exploited the idea of multi-views by simply augmenting the dataset with object rotations prior to feeding it to the neural network. The paper concludes that such an augmentation applied on the training set is crucial to improve the classification accuracy of the algorithm.

The idea behind the augmentation of the dataset with multiple viewpoints of an object is that we would like the network to learn the underlying features invariant to rotations. Sedaghat et al. [2] show that the classification accuracy can be further enhanced by providing additional information about the viewpoint and specifying a proper loss function in the training phase. The authors focus on two ideas: the first is to associate to each viewpoint of a sample the information about its rotation angle. The second idea is to define a new loss taking into account both the contribution of object and rotation classification. With this approach, they are able to force the algorithm to learn rotation features for each class individually. This distinction in viewpoints for different object classes is quite important, since different objects with the same angle of rotations may not share the same features. Indeed, the objective of the auxiliary task of rotation classification is not to learn shared rotation features among all classes, but to leverage the rotation information to improve the object classification performances. The authors applied this new idea to the VoxNet architecture (naming it the Orion network) and have been able to improve its classification accuracy.

3D convolutional architectures combined with voxel grid representation can be a computationally expensive task to perform. Beyond longer training times, this reflects also in a longer inference time. While the shallow architecture of VoxNet and Orion helps to maintain a lower inference time, it can be further improved by introducing a hybrid cascade architecture, as suggested in [3]. The main contribution of the paper is the introduction of an hybrid architecture, where different classification algorithms with increasing complexity are trained on the same task. At inference phase, the data will be processed firstly by simpler and faster architectures, and only the samples which produce classification results with high uncertainty are fed to the following architecture. The authors propose to use a shallow fully connected network as the first processing architecture. Samples rejected by it, will be processed by a shallow convolutional network. The third and last trained algorithm is an additional convolutional network, that takes as input the feature maps produced by the second stage. In addiction to this, the authors also propose an adaptive thresholding strategy to decide weather a result can be considered reliable or it needs to undergo the next classification stage. This approach allows to minimize inference time while maintaining a high accuracy. Exploiting the thresholding strategy, the authors have demonstrated to reach lower inference time than most of the state of the art algorithms present in the literature at the time of its publication.

III. Processing Pipeline and Learning Framework

In this section we will review the architectures implemented in this project. We will thus focus on VoxNet and Orion networks, underlying the basic details which allow them to be successful in the 3D classification task. Afterwards, we will introduce our implementation of a hybrid cascade network used to reduce inference time.

VoxNet architecture. VoxNet is a 3D convolutional neural network designed for object classification. It is composed by 2 convolutional layers followed by a fully connected layer, and finally the output layer (Fig. 1). Here we present the details of our VoxNet implementation:

- input data are voxel grids, with volume 32^3. Input objects are part of the ModelNet10 and ModelNet40 datasets, which will be briefly described in Sec. IV;
- the first convolutional layer (filter size 5, stride 2), convolves the input data in 32 feature maps, each one with a volume 14^3;
- the second convolutional layer (filter size 3, stride 1) produces 32 feature maps with volume 12^3;
- a max pooling layer (with dimension 2) transforms the 32 feature maps in shapes with volume 6^3;
- the obtained feature maps are flattened in a 1D array;
- this array will be processed by a fully connected layers, with 128 neurons;
- the output layer is composed by as much neurons as the number of categories in the dataset. The activation function of this layer is the ReLu function. Afterwards we apply a softmax function to the results, such that the output of the network can be interpreted as the probability associated to each class.

We applied dropout to each layer, and batch normalization after the convolutional layers. The activation function, common to each layer, is the Leaky ReLU. Convolutional layers were initialized with the method described in [6], whereas dense layers were initialized from a zero-mean Gaussian $\sigma = 0.01$. Each one of the hyperparameters of the network (apart from kernel sizes and strides) have been selected exploiting a

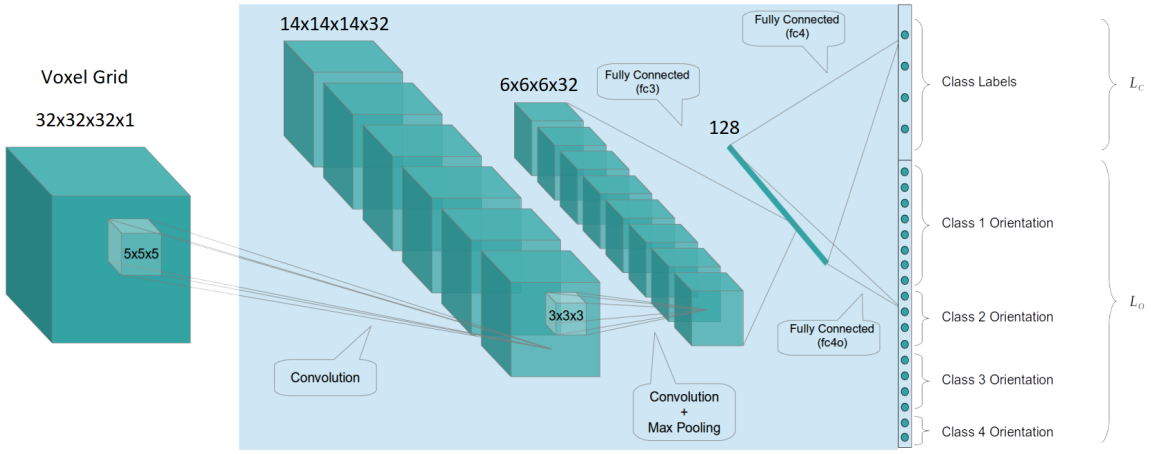


Fig. 1: Orion architecture. VoxNet actually shares the same structure, apart from the output layer. VoxNet only has as much neurons as the number of object categories, while Orion has additional neurons to classify the object rotation. Picture modified from [2]

Bayesian Optimization algorithm, as will be explained in detail in Sec. V.

Multi-view approach and Orion architecture. As stated in the previous sections, considering multi-view objects is essential to succeed in the 3D classification task. While VoxNet treats multi-views as an augmentation of the dataset, the Orion approach goes beyond and aims to train the algorithm also on rotation classification in order to improve the object classification accuracy. The Orion architecture shares completely its structure with the VoxNet one, apart from the output layer. Indeed, as can be seen in Fig. 1, the last layer of Orion is composed by the neurons for object classification (as for VoxNet) plus a certain number of neurons for each class. These neurons will serve for the orientation estimation task. We would like to emphasize the philosophy under this choice: disposing different output neurons for each object class allows to exploit the rotation classification task to improve category classification performances. The algorithm is discouraged to learn common features to predict the rotation label, instead it will derive different features for each class. Clearly, to force the algorithm to learn these features, we need to provide a specific loss in the training phase. The loss function proposed in [2] is defined as follows:

$$\mathcal{L} = (1-\gamma)\mathcal{L}_C + \gamma\mathcal{L}_O \tag{1}$$

where $\mathcal{L}_C$ and $\mathcal{L}_O$ are cross-entropy losses for the category and orientation classification tasks respectively. $\mathcal{L}_C$ is computed using only the first 10 or 40 output neurons, depending on the dataset (ModelNet10 or ModelNet40 respectively), while $\mathcal{L}_O$ is computed using all the remaining output neurons. γ is a free parameter which provides a trade-off between better accuracy on rotation or category classification.
In general each category may have a different number of rotation labels. This results in each class having a different number of neurons associated to its rotation prediction task. The reason behind this choice is the rotational invariance of some categories of objects. Indeed, in the dataset there are objects, like vases, which have a full rotational invariance, while other types of items are invariant only for rotations of $180°$ (e.g. screens) or $90°$. To avoid to confuse the algorithm in the training phase and let it try to learn differences in rotational-invariant objects, we need to take into account this fact. This is obtained by assigning a different number of rotation labels to each class. In Sec. IV we will describe how we processed the datasets to deal with this specific feature of the learning framework.

Voting procedure at inference phase. As described in [2], voting is a procedure which allows to improve the accuracy in the test set. It consists in processing through the network all the samples corresponding to different views of the same object, sum the outputs element-wise, and finally use this result to predict the object category. The rationale behind this idea is to leverage all available views of an object to make a prediction, thus correcting small errors that the network may commit for certain views. In a real application scenario of a deep learning architecture, this approach assumes that more than one view of an object is available. This technique typically results in an improvement of one or two percentage points in object classification accuracy when using all available views to make a single prediction.

Hybrid cascade architecture. Inspired from the ideas introduced in [3], we implemented our own hybrid architecture. In particular, it is composed by two networks (Fig. 2):

- a multilayer perceptron. Its input layer is composed of 512 neurons in order to fit a downsampled version of the voxelized 3D objects. The downsampling is obtained with a max pooling layer, producing a volumetric array with volume 8^3. Inputs are passed to the following layer composed of 256 neurons. The output layer is equivalent to the output layer of Orion, with the first

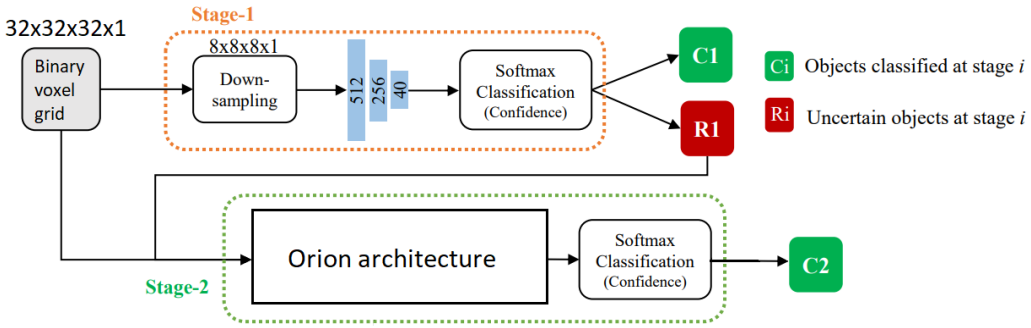


Fig. 2: Structure of the hybrid cascade architecture we implemented. Picture modified from [3]

neurons dedicated to category classification, followed by a number of neurons for each class to perform orientation estimation. This choice is due to the fact that it allows to obtain a slightly better accuracy, while the increase in inference time is negligible;

- the Orion architecture due to its success in 3d classification.

We choose to use only two networks because Orion achieves good enough classification results. Since our objective is to decrease inference time, a shallow fully connected network for the first stage should be a good choice. The two architectures are trained separately to perform the classification task. As discussed in the original paper, at inference phase the two networks are aggregated in the following fashion: we first use the multilayer perceptron to process a downsampled version of the input data. Then, we consider classified each sample if its classification probability is above a certain threshold. Samples that have been discarded are fed into the Orion network. In this way, we have Orion processing only "hard" samples, namely the ones that the faster multilayer perceptron cannot classify with high confidence.

Adaptive thresholding strategy. We will briefly review the adaptive thresholding strategy described in [3]. We compute the adaptive threshold θ_C^* for each class C as:

$$\theta_C^* = \arg\max_{\theta_C}(N_C^+(\theta_C))$$

$$\text{such that:} \quad \frac{N_C^+(\theta_C) - N_C^+(\phi_C^*)}{N_C^+(\theta_C) - N_C^+(\phi_C^*) + N_C^-(\theta_C)} \geq q$$

where $N_C^+(\theta_C)$ and $N_C^-(\theta_C)$ refers to the number of positive and negative samples of category C which have been classified by the multilayer perceptron with probability greater or equal than θ_C. ϕ_C^* refers to a specific threshold obtained in the following manner:

$$\phi_C^* = \arg\max_{\phi_C}(N_C^+(\phi_C))$$

$$\text{such that:} \quad \frac{N_C^+(\phi_C)}{N_C^+(\phi_C) + N_C^-(\phi_C)} = 1$$

With these definitions, $N_C^+(\phi_C^*)$ refers to the number of correctly classified samples of class C, in the case where none of the samples of that class is misclassified. In [3], the authors affirm that this thresholding method is able to achieve a high pass rate and a high precision simultaneously. q is a hyperparameter of the cascade architecture; it allows to obtain a trade off between a faster algorithm and a more accurate one. In our implementation, we exploit the validation set to obtain the adaptive thresholds for each class.

IV. Signals and Features

In this section, we will describe the dataset used to train the learning algorithms, the pre-processing implemented to reshape data in a suitable form, the generation of a validation set and the augmentation performed on the samples.

Datasets. We trained our networks on two different datasets: ModelNet10 and ModelNet40. The datasets have been introduced by Wu et. al. [7]. ModelNet10 consists in $\sim$ 4900 3D triangular meshes split in 10 categories. ModelNet40 contains $\sim$ 12300 samples, divided in 40 different categories. We note that, while ModelNet10 containes aligned objects, the same does not hold for ModelNet40. Since the alignment is an essential feature when dealing with multiple views of 3D objects, Sedaghat et. al. [2], introduced a manually aligned version of the dataset. We used this version to our purpose.

Pre-processing. The datasets contains 3D meshes, which are represented in the .off file format. To adapt the samples to the requirements of our network, we need to convert the meshes in voxel grids. To do so, we exploited a Matlab implementation of the voxelization task provided by Sedaghat et. al. [2], available at the linked *GitHub repository*. The code provided navigates through the folders of the dataset to convert each .off file in a .mat file containing a 3D voxel grid in the form of a $V \cdot V \cdot V$ array. The conversion itself is handled by a C++ function, which converts the 3D mesh to a binary voxel grid, where the value of 1 indicates an occupied voxel and 0 indicates an empty space. The voxel grids are "filled" in the sense that the internal volume of each object is filled with occupied voxels. We adapted the code provided to our needs, setting some parameters such as voxel grid volume (set to 24^3) and padding (set to 4 for each dimension), thus obtaining 32^3-volumetric voxel grids. Another task the cited Matlab code performs is the augmentation of the dataset with multi-views

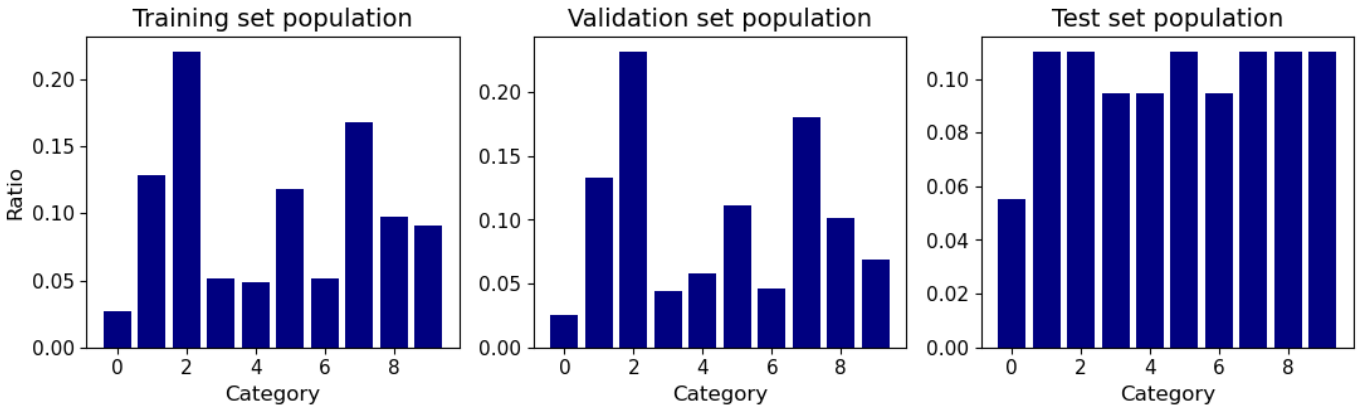


Fig. 3: Dataset population distribution

of the same object. With an appropriate routine, the script rotates each mesh prior to convert it, and then save the voxel grids obtained from the rotations of the object in different Matlab files. Following the recipe given in [2], we set the algorithm to compute multi views at intervals of 30° in the horizontal plane, till a full rotation is completed. This choice results in 12 views per object.

After converting the dataset objects into voxel grids, we implemented our own code to write each dataset into a HDF5 file. The advantage of this format is its ability to store heterogeneous data grouped in different datasets, and it represent a valuable tool to navigate through the dataset in a fast and memory-efficient way. Apart from the voxelized objects, we inserted in the HDF5 file also the information about the category label and the rotation label. Regarding the rotation labels, we saved two distinct dataframes: the first with the labels representing the true rotation angle, regardless the rotational invariance of each object, while the second incorporates this additional information and modifies the labels consequently. In particular, we redefined the rotation labels in an adequate range to describe the rotational invariance property of each object category.

Validation set. Both datasets come with a split-up between training and test data. Approximately 80% of samples are intended for training. We further partitioned this training set to obtain the actual training set with 80% of the original samples, and a validation set with the remaining 20%. The validation set is used not only to optimize hyper-parameters of the learning framework, but also to implement the early stopping technique. To accomplish both tasks, we need a validation set which is more representative as possible of the test set. To satisfy this requirement, samples to be inserted in the validation set are chosen randomly, but with a particular attention to not separate different views of the same object among training and validation sets.

Fig. 3 shows the population distribution of the different classes in the dataset. We note that the test set comes with a different sample distribution among classes compared to the training set. This can potentially result in an additional decrease of the classification accuracy in the test set.

Data augmentation Before passing training samples to the training scheme, we may want to augment them with some randomized transformations. We chose three augmentation steps suited for 3D objects, to be applied in sequence:

- Mirroring: with probability 0.3, samples are flipped along the $\hat{x}$ direction, the $\hat{y}$ direction, or left as they are. This transformation is actually applied only in the case where we do not classify rotations, namely when training the VoxNet architecture;
- Shifting: in each of the three dimensions, we perform a random shift of the object by a maximum of 2 voxels. In this way, we can maintain a minimum padding of 2 voxels;
- Addition of Pepper noise: with a certain probability, we randomly flip filled voxels to obtain empty ones.

V. Hyper-Parameters Optimization

In this section we will describe the learning procedure adopted to train the various architectures and to obtain good classification performances. To achieve the best results from the training procedure, we need to carefully tune the hyper-parameters of the network. For the hyper-parameter optimization we exploited the Optuna framework. In particular, to efficiently search in the parameter space, we used the Tree-structured Parzen Estimator [8], which is a sampler in the class of the Bayesian Optimization methods. To further speed up the optimization, we used only ModelNet10 dataset: since the structure of the network is the same for both the datasets, we assume there are no important differences in the optimzed parameter configuration. We used the validation loss as the metric to investigate which parameter configuration is optimal. We chose the validation loss because it is a ”soft” metric which carries more information compared to the hard choice performed in computing the accuracy.

VoxNet training. As described in Sec. III, the output layer of VoxNet consists in as many neurons as there are categories in the dataset, resulting in 10 output neurons for ModelNet10, and 40 neurons for ModelNet40. The output is processed

$\mathcal{L}$	Optimizer	η	γ
0.182	Adam	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$
0.193	SGD	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$
0.227	Adam	$5 \cdot 10^{-4}$	$1.3 \cdot 10^{-4}$
0.246	SGD	$7 \cdot 10^{-4}$	$4.4 \cdot 10^{-5}$

(a) VoxNet on MN10. Gradient Descent hyper-parameters optimization

$\mathcal{L}$	p dropout	ν
0.171	0.41	0.02
0.180	0.30	0.13
0.192	0.35	0.24
0.199	0.18	0.19

(b) VoxNet on MN10. Network hyper-parameters optimization

$\mathcal{L}$	batch size
0.171	32
0.182	100
0.183	281
0.187	17

(c) VoxNet on MN10 - Batch size optimization

$\mathcal{L}$	γ
0.493	0.49
0.508	0.19
0.515	0.04
0.519	0.83

(d) Orion on MN40 - γ loss parameter optimization

$\mathcal{L}$	Optimizer	η	λ
0.48	Adam	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$
0.67	Adam	$1 \cdot 10^{-2}$	$1 \cdot 10^{-3}$
0.68	SGD	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$
0.72	SGD	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$

(e) MLP on MN10. Gradient Descent hyper-parameters optimization

$\mathcal{L}$	p dropout
0.476	0.3
0.484	0.4
0.486	0.2
0.491	0.5

(f) MLP on MN10. Dropout probability optimization

TABLE 1: Some results of hyper-parameters optimization on our architectures. Here, $\mathcal{L}_C$ refers to the category validation loss.

through a softmax function to obtain a set of probabilities of the sample being an instance of the different classes. The category labels are represented with a one-hot encoding. To train the algorithm we used the cross-entropy loss, which is a common choice in case of multi-class classification problems. In the following, we will describe the optimization of the hyper-parameters of the model. It is worth to mention that we did not try to optimize the network structure, such as its number of layers, feature maps, kernel sizes, etc. This choice is due to two reasons: first, we would like to implement a network reasonably similar to the one implemented in [1] to obtain a direct comparison with it. Second, the authors declare they performed an extensive search over hundreds of 3D CNN architectures, so the architecture of VoxNet should be already optimized.

We divided the hyper-parameters in three sets. The first set contains the hyper-parameters pertaining the gradient descent strategy, namely the optimizer, the learning rate η and the λ parameter for L_2 regularization. The second set is dedicated to the parameters of the network architecture, such as dropout probability p and the parameter ν of the Leaky ReLU activation function. The last set is dedicated to the optimization of the batch size.

Tab. 1a, 1b, 1c show some results obtained from the hyper-parameters optimization process. Let's comment briefly these results. Regarding the first hyper-parameters set, Maturana and Scherer [1] choice fall on the SGD optimization method, with momentum 0.9. The second row of Tab 1a refers to the configuration adopted in the cited work. We investigated if Adam could be a better choice for the training. While for the parameters η and λ our results are in agreement with the authors, we find that Adam actually improves the results compared to SGD. We will discuss further the choice of the optimizer in the next section. Regarding the second set of hyperparameters, we find that a dropout probability of 0.4 minimizes the validation loss. On the other hand, the Leaky ReLU parameter ν appears to have low relevance on the algorithm performance. The same holds for the batch size. Thus, we fix these two hyper-parameters as Maturana and Scherer did: $\nu = 0.1$, batch size $= 32$.

Orion training. In Sec. III we explained how the output layer of Orion is defined, and the loss used to train the algorithm. The only detail we wish to specify is that all rotation-devised neurons are involved in the computation of the loss simultaneously. Thus, once the network has elaborated the input signals and returned its outputs, we apply two softmax functions: the first to first 10 or 40 category-devised neurons, the second to all the remaining rotation-devised ones. The rotation labels are defined with one-hot encoding, with the value 1 positioned in the place corresponding to the correct view of the correct object category. In this way we force all the neurons of the wrong category to be 0, thus enforcing the category prediction ability.

As the Orion architecture shares almost completely its structure with the VoxNet one, we set the hyper-parameters of the model as the results obtained for VoxNet suggests. The only free parameter left is γ, defined in the loss function in Eq. 1. Sedaghat et. al. [2] found that 0.5 is a good value for it. Our results, partially reported in Tab. 1d, supports this finding. In particular, for high values of γ, the algorithm is forced to learn primarily the rotation estimation task, thus ignoring its main assignment. On the other hand, low values of γ makes the algorithm focus mainly on object classification, thus reducing to the VoxNet network.

Multilayer Perceptron training. The Multilayer Perceptron has the same output layer as Orion (neurons dedicated to category classification followed by neurons that perform orientation estimation); for this reason we use the loss defined in Eq. 1 with the parameter γ set to 0.5. During our experimentation with the network, various layer sizes were investigated, ultimately leading to our decision to maintain an architecture identical to the one described in [3], with 256 neurons in the hidden layer. For this architecture we

Feature	VoxNet		Orion	
	MN10	MN40	MN10	MN40
Baseline	0.174	0.477	0.149	0.483
SGD	0.209	0.468	0.171	0.428
Data augmentation	0.182	0.505	0.152	0.487
No Batch normalization	0.179	0.489	0.151	0.475
No rotational invariance	-	-	0.148	0.462

(a) Optimizer test. Values are the (category) validation losses. Baseline refers to set of following features: optimizer: Adam, no data augmentation, batch normalization, consideration of rotational invariance

/	Orion	
	MN10	MN40
With rotational invariance	0.269	0.605
Without rotational invariance	0.516	1.294

(b) Rotation validation loss with and without considering rotational invariance

TABLE 2: Tests conducted on VoxNet and Orion

/	MN10			MN40		
	cat. val. loss	global acc.	avg acc.	cat val loss	global acc	avg acc
VoxNet - our implementation	0.174	90.5	89.6	0.468	86.3	81.9
Orion - our implementations	0.149	91.4	90.8	0.428	86.2	82.1
VoxNet [1]	-	-	92	-	-	83
Orion [2]	-	93.8	-	-	88.2	-

TABLE 3: VoxNet and Orion - final results. Cat. val. loss, global acc. and avg acc. refer respectively to the category validation loss, micro and macro accuracy computed on the test set. We also report the results stated in the literature

explored different optimization algorithms, their parameters and the values for the dropout probability. The results of this investigation are summarized in Tab. 1e and 1f. Similar to the VoxNet architecture, we observe that Adam with learning rate $\eta = 0.001$ and L_2 regularization $\lambda = 0.001$ leads to better outcomes. For the dropout, we have set $p = 0.3$. These values for the hyper-parameters were confirmed also with the ModelNet40 dataset.

VI. Results

Once defined the learning architectures and optimized their hyper-parameters, we can train the algorithms on ModelNet10 and ModelNet40 to evaluate their performances. This section will be organized in three different parts: first we will conduct some tests on the VoxNet and Orion architectures to evaluate the effectiveness of some features we implemented, such as batch normalization, data augmentation, the control on objects rotational invariance and a deeper investigation between the choice of Adam or SGD as the optimizer. In the second part of this section, we will provide the final results obtained with our VoxNet and Orion implementations. In the last part, we will discuss the effectiveness of our hybrid cascade architecture.

Tests on VoxNet and Orion In the following discussion, the algorithms are trained with fixed specifications when not otherwise stated. This specifications are: consideration of rotational invariance, Adam optimizer, no data augmentation and batch normalization. We will vary this properties individually, to assess the effectiveness of each of them. Tab. 2 summarizes the results of the tests. We will now briefly discuss each of them.

In Sec. V, we conducted an Optuna study on VoxNet using ModelNet10, and assessed that Adam is able to achieve lower validation losses. We conducted here a more in-depth test since the authors of [1] declare that SGD is a better optimizer. What we found is that actually SGD obtains better results when the networks are trained on ModelNet40. We take note of this fact, and to obtain the final results we will use SGD to train both the architectures in this particular dataset.

In Sec. IV we described which data augmentation steps we defined. Tab. 2a shows how data augmentation is unable to help the algorithm to learn the classification task. This is not unexpected: due to the pre-processing performed on the datasets (Sec. IV) we have that objects in the test set are highly standardized. Indeed: they are always in the same position in the cubic box, and they rarely (if not never) are affected by some form of noise. Due to this reasons, the learning algorithms actually perform better when trained on a dataset with no data augmentation.

Regarding the effectiveness of batch normalization, we found that generally it does not have a significant improve in the accuracy of the model. This comes as a confirmation of what we expected: our networks are shallow ones, with only two convolutional layers. We know that batch normalization has a great impact in training very deep model, while in shallow ones the effects that batch normalization tries to neutralized, e.g. internal covariate shift, are negligible. Even if batch normalization effectiveness in our models is questionable, in our final results we will use it in the definition of the algorithms, since in some cases it produces a small improvement at the cost of only two learnable parameters per layer.

Finally, we studied the impact of the consideration of rotational invariance in the Orion learning framework. To briefly recap what was mentioned in Sec. III, some of the object category present a rotational symmetry under certain degree of rotation. As suggested in [2], we introduced a modification

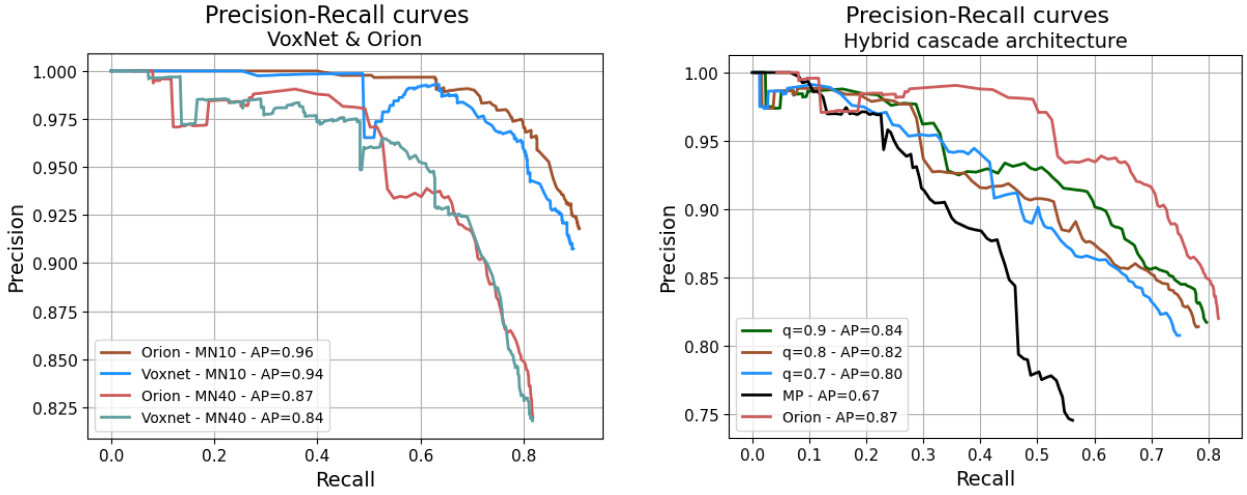
	$t(ms)$	accuracy	
		MN10	MN40
VoxNet	0.70	90.5	86.3
Orion	0.73	91.4	86.2
Multilayer Perceptron	0.28	84.7	69.4

(a) Mean inference time and accuracy for different models

q	MN10		MN40	
	accuracy	$t(ms)$	accuracy	$t(ms)$
0.6	88.5	0.29	74.5	0.40
0.7	88.6	0.30	79.2	0.51
0.8	88.8	0.37	82.2	0.60
0.9	89.6	0.43	84.8	0.68

(b) Mean inference time and accuracy under different threshold values on MN10 and MN40.

TABLE 4: Mean inference time for different architectures and threshold values



(a)

(b) Results on ModelNet40

Fig. 4: Precision - Recall curves. Results obtained on the test set.

both in the output layer of the Orion architecture and in the rotation labels to account for this symmetry. Here, we interrogate the learning model to evaluate the impact of this correction. Despite what we think is a theoretically valid argument, Tab. 2a shows that in practice the effectiveness of these corrections is highly questionable. Indeed, looking at the category validation losses in the datasets, we find that they are lower without considering the object rotational invariance, especially in the case of ModelNet40. On the other hand, Tab. 2b shows how great is the impact of considering or not rotational invariance in rotation estimation. To provide a more meaningful result, we report on ModelNet40 validation accuracy in rotation estimation of 82.3% considering rotational invariance, which decreases to 60.1% if we don't consider it. We also want to emphasize the comparison between the second and fourth row of Tab. 2a, which refer respectively to SGD + rotational invariance and Adam + no rotational invariance. In this case, when trained on ModelNet40, the best model appear to be SGD + rotational invariance. In conclusion, we infer that the effectiveness of rotational invariance corrections in the training model is unclear.

Looking at the different columns of Tab. 2a, we can select the best choices for each combination of learning architecture and dataset. To summarize the results, we obtain that the best choice for networks trained on ModelNet10 is to use the Adam optimizer, with batch normalization and without data augmentation at the training phase. On the other hand, when the networks are trained on ModelNet40 it is better to use the SGD optimizer, while the other specifications remain unchanged. In both cases, we will consider rotational invariance in the definitions of our datasets and networks.

VoxNet and Orion results Considering the implementation choices described above, we trained our best models. The results are summarized in Tab. 3. We can finally appreciate the effectiveness of the choices introduced in [2] to define the Orion architecture. While for ModelNet10 we obtain that Orion introduce a significant improvement, we can not say the same in the case of ModelNet40. Indeed, the accuracies obtained in the test set for VoxNet and Orion appear to be very similar. On the other hand, we note that the category loss computed on the validation set is significantly smaller for Orion. This suggest that Orion should be in any case a better architecture to predict object category. The very small differences in the accuracy may be caused by the specific test set used, while in other sets we may appreciate a bigger difference. It is important to note that we mainly performed the optimization of the hyper-parameters on ModelNet10, and only did some checks on ModelNet40 to confirm the success

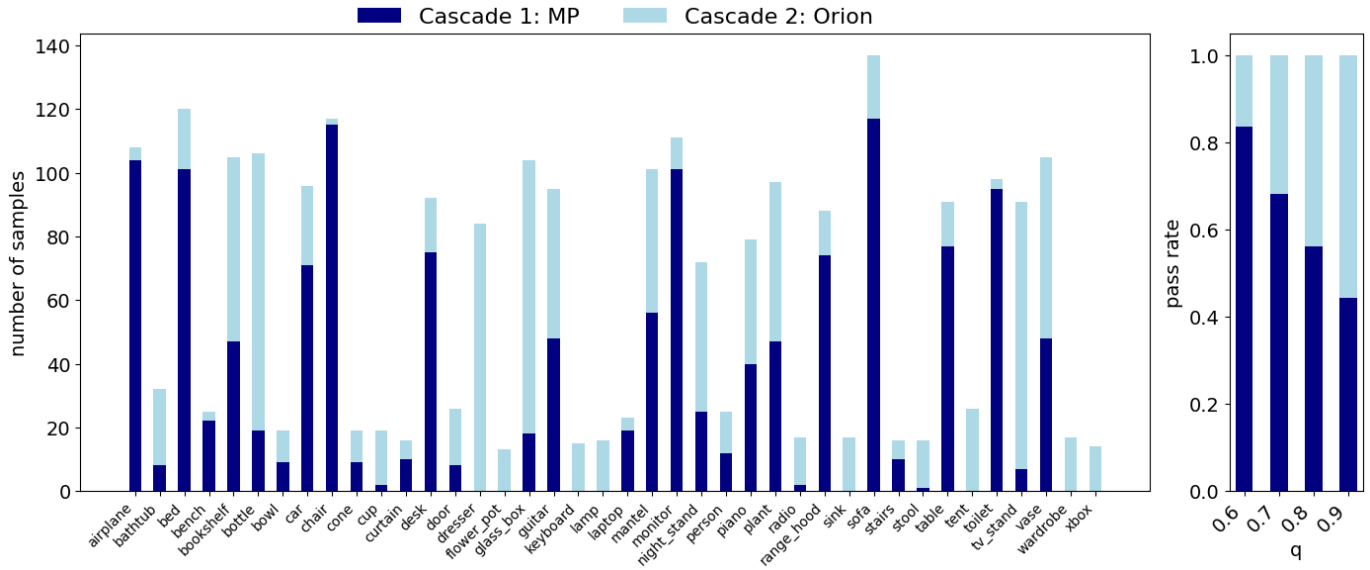


Fig. 5: The plot on the left shows the distribution of the number of samples classified by MP and Orion per category for threshold value $q = 0.8$. On the right are shown the pass rates for different threshold values on ModelNet40

of the optimization process. Overall, we can state that Orion produce a slight improvement in the 3D classification task compared to VoxNet. This results confirms that forcing the architecture to explicitly learn rotation features for each class can effectively enhance the object classification accuracy.

We also compared our trained models to the ones reported in the literature. We found that the accuracy obtained by our models is in general two percentage points below the one obtained in the literature.

Fig. 4a shows the precision-recall curves for the models we trained. We decided to show this type of analysis since it is more suited in cases with a high class imbalance (Fig. 3). The results obtained with ModelNet10 are obviously better than the ones achieved with ModelNet40. What is interesting to note is the difference among the two architectures. Reflecting the results obtained in Tab. 3, we observe that on the ModelNet10 dataset there is an appreciable difference in VoxNet and Orion results, where Orion obtains a higher AP. On the other hand, we note that the performances of VoxNet and Orion on ModelNet40 are almost superimposable. This is another confirmation that on ModelNet40, our implementation of the Orion architecture is unable to improve the performances of VoxNet.

Hybrid cascade architecture This architecture was introduced in order to optimize the mean inference time. As illustrated in Tab. 4a, the Multilayer Perceptron (MP) has notably reduced inference time compared to VoxNet and Orion, which show similar performance.

In Sec. III we discussed the adaptive threshold strategy, the trade off between speed and accuracy and how it is linked to the choice of the parameter q. We know that Orion has higher accuracy compared to MLP but it is also much slower. This trade-off between the two properties can be observed in Tab. 4b where we present results for different values of the threshold. Fig. 5 shows the distribution per category of the pass rate for a threshold value $q = 0.8$. It is evident that certain categories exhibit high θ_c values, indicating that the majority of them are efficiently classified by the MLP algorithm. On the other hand, more challenging categories, which often require additional examination, are predominantly directed to Orion.

In Fig. 4b are shown the Precision-recall curves for Orion, MLP and some Hybrid cascade architectures obtained by varying the value of the hyper-parameter q. As expected, we can see that Orion performs better then MLP and we have the results form the cascade model in between them. Larger q values bring the curve closer to Orion, while smaller q values make it approach MLP.

VII. Concluding Remarks

In this project, we delved into the field of 3D object recognition. In particular, we implemented two convolutional neural networks introduced in recent years: VoxNet [1] and Orion [2]. We trained them on two different voxelized datasets obtained from ModelNet10 and ModelNet40, to investigate the magnitude of the improvements introduced with the Orion architecture. Tab. 3 summarizes the obtained results. Our trained models performs only slightly worse than the ones reported in the literature. From our study, we inferred that Orion is able to improve the performances of VoxNet in the ModelNet10 dataset, while on ModelNet40 they obtain similar results. Since we only performed hyper-parameter optimization on ModelNet10 due to the constraint of computational resources at our disposal, it is possible that a finer hyper-parameter optimization conducted on ModelNet40 may improve the performances of Orion when trained on this dataset. In addiction, regarding the Orion architecture,

we studied the impact that rotational invariance has on the object classification task. We found that considering or not rotational invariance does not affect the accuracy on category classification. Ultimately, we found that an improvement in the rotation estimation task does not result in an enhancement of the category classification task. This partially reflects the result of the performances of the architectures on the ModelNet40 dataset.
In addiction, inspired by [3], we implemented a hybrid cascade architecture. It is able to achieve an improvement in inference time while maintaining a good accuracy. Tab. 4 summarized the results, showing how the cascade architecture results in a trade-off among a fast inference process and a good classification accuracy.
The comparison of our models with existing literature indicates potential for further optimization in our architecture. A possible way to improve the classification performance could be to insert additional convolutional layer in the Orion implementation, or to design an optimized set of parameters to weight distinctly the loss computed in each category. Despite the changes to the Orion architecture that could be made to improve its performances, the great flexibility of the cascade network allows to simply replace a new trained model with the actual Orion model, without the needs of additional computations.

VIII. Additional notes

While in our project we did not experiment with different and advances network architectures, we were still able to draw many notions from it. The notions we learned ranges from an acquired familiarity with 3D representation formats, to the usage of the torch frameworks and it's integration with other platforms, such as Weight & Biases to visualize and store our training runs, and Optuma to perform efficient hyper-parameter optimization. We were able to acquire the basic notions of the workflow in the deep learning field, from the network definitions, to its optimization and the extraction of relevant results. In addiction, we had the possibility to improve in other general tasks, such as literature navigation and the redaction of a proper article.
The main challenge to our work was the bounded accessibility to computational resources such as GPUs. Despite Google Colab being a great tool to experiment in the deep learning field for free, it is also being limited in the amount of computational resources provided. This characteristic forced us to develop a good scheduling of the training tests to perform. The main area of the project limited by this factor has been the parameter optimization.

Here we will provide a list of each individual contribution:
Peri Andrea:

- pre-processing of the datasets;
- optimization of the VoxNet and Orion hyper-parameters;
- realization of the final tests conducted with the VoxNet and Orion architecture;
- first drafting of the initial part of this report.

Mojsovska Marija:

- definition of the VoxNet and Orion architecture, implementation of the training code;
- definition and optimization of the multilayer perceptron;
- implementation of the adaptive thresholding strategy;
- extraction of the results obtained from the hybrid cascade architecture.

We want to conclude asserting that, despite our diversified contributions, we worked together in all aspects of the project, with ongoing discussions and exchange of information have been in place during the whole duration of the presented work.

References

[1] Daniel Maturana and Sebastian 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. DOI: 10.1109/IROS.2015.7353481.

[2] Nima Sedaghat et al. *Orientation-boosted Voxel Nets for 3D Object Recognition*. Jan. 2017. DOI: 10.5244/C.31.97.

[3] Hui Cao et al. *Fast Hybrid Cascade for Voxel-based 3D Object Classification*. Nov. 2020.

[4] R. Charles et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2017, pp. 77–85. DOI: 10.1109/CVPR.2017.16.

[5] Seyed Saber Mohammadi, Yiming Wang, and Alessio Del Bue. "Pointview-GCN: 3D Shape Classification With Multi-View Point Clouds". In: *2021 IEEE International Conference on Image Processing (ICIP)*. 2021, pp. 3103–3107. DOI: 10.1109/ICIP42928.2021.9506426.

[6] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *IEEE International Conference on Computer Vision (ICCV 2015)* 1502 (Feb. 2015). DOI: 10.1109/ICCV.2015.123.

[7] Zhirong Wu et al. "3D ShapeNets: A deep representation for volumetric shapes". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2015, pp. 1912–1920. DOI: 10.1109/CVPR.2015.7298801.

[8] Shuhei Watanabe. *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. 2023. arXiv: 2304.11127.