The importance of orientation in 3D Object Classification

Davide Peron[†] and Emanuele Vitetta[†]

***Abstract*—Recent advancements in 3D object recognition achieved through convolutional neural networks are promising. In this manuscript, two key convolutional neural networks, namely ORION and RotationNet, designed for object classification are studied. ORION operates on 3D input data, whereas RotationNet processes 2D inputs; both solutions address the same recognition challenge with orientation playing a pivotal role in enhancing their performance. In fact, objects are analysed from different viewpoints; this allows to achieve a better feature comprehension. Our analysis underscores the fundamental importance of object orientation in training. The considered architectures leverage orientation information to significantly improve classification accuracy. An evaluation across diverse 3D data sources is provided.**

***Index Terms*—3D Object Classification, Convolutional Neural Networks, ORION, RotationNet, ModelNet.**

I. Introduction

In the realm of modern technology, the prevalence of devices leveraging 3D models is continuously expanding. Sensors capable of capturing 3D point clouds have gained widespread adoption due to cost reductions and have improved accuracy. As a consequence, the ability to swiftly analyse complex models has become increasingly vital for a myriad of applications; these include industrial robotics for environment sensing, *Simultaneous Localisation and Mapping* (SLAM), and vision systems for autonomous vehicles.

The processing of 3D objects has required the development of novel neural network architectures, poised to deliver good performance in terms of execution time and precision. *Convolutional neural networks* (CNNs) have been instrumental in this context. Nonetheless, the potential to generalise outcomes occasionally falls short. In fact, the inherent structure of these networks tends to capture only a subset of the key features required for classification across a broad spectrum of objects.

During the evolution of the above mentioned tools, the significance of object orientation in enhancing performance has emerged. This manuscript aims at reviewing various neural network architectures that undertake object classification. All of them exploit object orientation to improve their performance in terms of accuracy. Both 2D and 3D CNNs [1] [2] are considered, highlighting their differences in terms of architecture, complexity, and training choices.

In order to make all our results comparable with a large number of CNNs, we have trained the reviewed models both on ModelNet10 and ModelNet40 datasets. Our primary goal was to achieve the same results shown in specific manuscripts [1] [2] and understand if it was possible to improve upon them.

The remaining part of this manuscript is organized as follows. Section III delves into the critical preprocessing phase, essential to achieve the desired outcome. Subsequently, Sections IV and V explore both the input and architectural choices we made. Finally, in Section VI we illustrated our findings.

II. Related Work

The integration of 3D CNNs has initially appeared in the domain of video analysis. Notably, the potential of 3D CNNs extends beyond videos and encompasses other three-dimensional inputs, such as Voxel grids, that represent the focus of our present work.

In the related literature, the contributions provided by Wu *et al.* [3] and Maturana & Sherer [4], namely 3D ShapeNets and VoxNet, stand in close relation to our work. In particular, Wu *et al.* propose a *Deep Belief Network* (DBN) able to capture the essence of geometric 3D shapes through a probability distribution of binary variables on a 3D Voxel grid. Their methodology finds application in shape completion from depth maps, a concept that leads to the introduction of the ModelNet dataset. The VoxNet [4], instead, introduces a simple yet impactful CNN architecture. It operates on voxel grids akin to the approach of Wu *et al.* [3]. Notably, both of these works leverage data augmentation via object rotation, fostering an invariant feature representation learning. However, unlike the networks we consider, their architectures do not aim at predicting object orientation; they solely output class labels.

Sedaghat *et al.* [1] introduced a noteworthy breakthrough by integrating orientation considerations through their ORION framework. This architecture, built upon VoxNet, fundamentally focuses on predicting the object class. However, as a secondary objective, the network concurrently tries to estimate the object's pose. Consequently, during the training phase of the neural network, the loss function takes into account both the object's class and its pose estimation. This strategic augmentation results in discernible enhancements in classification accuracy.

Another relevant work by Kanezaki *et al.* [2], [5], namely RotationNet, highlights the importance of addressing object orientation in network training, capitalising on the idea that an object's intrinsic features should remain invariant with respect to variations in orientation. The main difference with respect to the other two CNNs mentioned above is represented by the

[†]Department of Information Engineering, University of Padova,
email: davide.peron3@studenti.unipd.it,
emanuele.vitetta@studenti.unipd.it

input passed to the NN. In fact, in the case of RotationNet, the input is composed of multiple 2D images representing different viewpoints of the 3D object. Hence, the main idea is to use 2D features to extract information about 3D models. This requires an intermediate preprocessing step. Furthermore, this solution involves a 2D CNN. Such a structure is widely used for image classification; hence, its optimization is slightly easier, due to the presence of a number of results in the existing literature.

III. Processing Pipeline

Our work follows a precise strategy in order to make a fair comparison between the ORION and RotationNet architectures, highlighting the main differences in terms of training and performance. First of all, the considered architectures have been trained using ModelNet10 and ModelNet40 [3] [6]; both of them are a comprehensive clean collection of 3D CAD models for objects. The properties of each one of them are listed in Tab. 1. Furthermore, some objects belonging to the dataset are shown in Fig. 1.

Dataset	Classes	Train Samples	Test samples
ModelNet10	10	3991	908
ModelNet40	40	9843	2468

TABLE 1: Splitting of original datasets.

The training strategy has been different for ORION and RotationNet, in terms of both the employed input and the considered optimisation methods. In the first case, data augmentation has been performed: for each of the 3D models, different copies with a new orientation have been generated. All the new samples have been labelled with both their class and pose (the number of pose classes varies among different models). At this point, the augmented datasets have been converted into voxel grid format. Note also that ORION generates not only the estimated class but also the estimated pose of each 3D model.

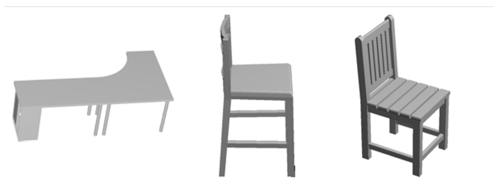


Fig. 1: Example of some 3D models. Note the similarity between the central model and the one on its right (even if they belong to different classes). Objects are observed from different viewpoints.

In the second case, we employed a dataset [7] already given in the form of 2D images (we did not perform the conversion of 3D to 2D objects, in order to have the same benchmark values). Note that RotationNet requires 2D inputs. In this case, for each 3D model, a set of 12 or 20 images representing such a model from different viewpoints is sequentially fed into the NN. Each image is labelled by the object class only, whereas the viewpoint characterising it is considered as a latent variable. This clearly reduces the preprocessing effort.

After feeding each NNs with its correct input, we performed some tests in order to achieve the best possible accuracy (that may differ from the one shown on the original papers). All the tests for ORION have been performed by trying different weight initialisations, activation functions and optimisation methods. Fewer tests have been performed on RotationNet because of its large complexity.

IV. Signals and Features

In this section an accurate description of how the input of each of the considered NNs is obtained. As explained in this section and in the following ones, data preprocessing highly influences the training phase and, hence, the final accuracy achieved by a given network.

A. ORION

In our work, we used a version of ModelNet (for both 10 and 40 classes) whose elements have been manually pre-aligned with the same orientation (thanks to the work of [8]). This ensures coherence among different models and classes.

The following steps have been performed on each 3D model to generate the data feeding ORION:

1) A number of copies with a different orientation is generated for each CAD model (in the original paper [1] this can be identified, for example, as Az$\times$12, where 12 stands for the different rotations and Az stands for Azimuth rotation around the z-axis).
2) All the elements obtained in the previous step are converted in binary voxel grids of size $32 \times 32 \times 32$, where the object occupies the $28 \times 28 \times 28$ central units. The remaining voxels are obtained by padding.
3) Each voxel grid is associated with two labels: one regarding its class, the other one its orientation. The number of classes selected for the orientation depends on two factors: the experiment and the symmetry of the object. In fact, we tend to use a smaller number of rotation classes for objects that are symmetric around some axis. Furthermore, when we have many similar classes, we use fewer rotations, in order to make results more consistent.

Another fundamental step is represented by the generation of the validation set. We modified the code [9] given by the authors of [1] to generate a validation set whose distribution is similar to the test one. Such a change made the validation loss a more accurate metric for model selection. Finally, we decided to divide the train-validation-test set roughly as 60-20-20.

B. RotationNet

The data preprocessing required for this architecture is more complicated. To limit the computation time and obtain results comparable with those provided in the technical literature, we used the dataset of 2D images provided by the authors of

[2]. Such images are generated on the basis of all the models belonging both to ModelNet10 and ModelNet40, and taking into account a set of viewpoints of such models. As shown in Fig. 3, the three cases described below are considered.

Case (i) - In this setup, the z-axis is selected as the rotation axis, defining an upright orientation. Each viewpoint is placed at intervals, regularly spaced of an angle θ around such an axis. In our work, $\theta = 30°$ has been selected; this choice leads to 12 distinct views ($M = 12$) for this object. Moreover, it is assumed that the "view $m + 1$" is obtained by rotating the "view m" by the angle θ around the z-axis. Note that the view obtained by rotating "view M" by the angle θ corresponds to "view 1". The sequence of input images is assumed to be consistent with respect to a certain direction of rotation in the training phase. For instance, if v_i is m ($m < M$), then v_{i+1} is $m+1$. Thus, the number of candidates for all the viewpoint variables $\{v_i; i = 1, 2, ..., M\}$ is M.

Case (ii) - In this setup, no assumption is made about the upright orientation. Some virtual cameras are placed on the $M = 20$ vertices of a dodecahedron. This choice is motivated by the fact that a dodecahedron is the regular polyhedron having the largest number of vertices, so that viewpoints can be completely equally distributed in 3D space encompassing the object. Differently from case (i), there is no unique rotation direction, but there are three different patterns of rotation from a certain view, since three edges are connected to each vertex of a dodecahedron. Hence, the number of candidates for all the viewpoint variables $\{v\}_{i=1}^{M}$ is $3M = 60$. An example of the generated viewpoints in this case is shown in Fig. 2. In network training, different rotations for ModelNet10 and ModelNet40 have been used.

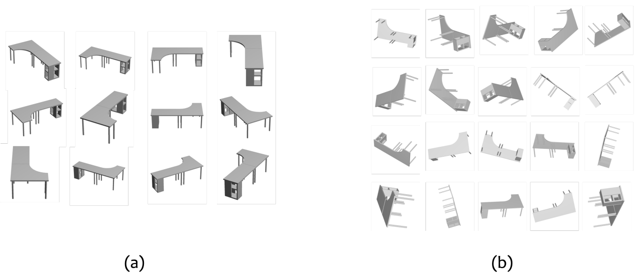


Fig. 2: Example of viewpoints generated in various cases: (a) 12 views of case (i); (b) 20 views of case (ii). The order according to which the images are displayed is not relevant.

Case (iii) - This setup is only taken into account by the authors of [2]. We did not consider this set of viewpoints because of the large computational effort it required.

It is also important to mention that the choice of the validation set depends on the considered case. In fact, in case (i), since the NN is trained using a reduced dataset, the test set is also used as validation set. This is not the best possible choice, but generating a real validation set would further reduce the size of the training. The small size of the training set also causes other problems (e.g., overfitting). On the other hand, in case (ii), the validation set has been generated exactly as for ORION. This implies that a given element pertains to the identical set (either training or validation) within both datasets.

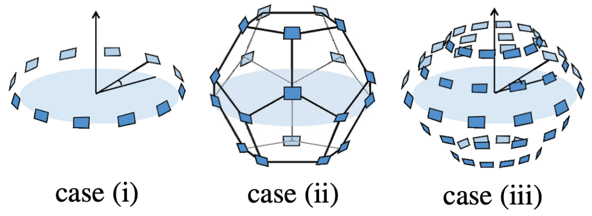


Fig. 3: Different sets of viewpoints considered for each 3D model. The target object is fixed at the centre of each shape. Figure taken from [2].

Finally we point out that there is a slight difference between the datasets generated for ModelNet10 and those for ModelNet40. Indeed, for each 3D model, the considered viewpoint for the two databases is different. Such a choice is made to achieve a better accuracy for a given dataset, as highlighted in [2].

V. Learning Framework

In this section, the architectures of the considered NNs are explained in detail. Moreover, we comment on their training and some various technical choices we made.

A. ORION

The architecture of this NN is shown in Fig. 4. In our test, an extended version of ORION, made of 4 convolutional layers, has been developed. Filters with kernel size $3 \times 3 \times 3$ characterize each of them. A pooling layer follows the last convolutional layer. In the last portion we have some fully connected layers; in particular, the very last layer is made of two fully connected ones working in parallel. A different task is assigned to each of them: the first one returns the object class, which is the only output of the NN we consider. Actually, ORION also provides the object pose. This value is not relevant for performance comparison with other architectures, but it is fundamental during the training phase. In fact, assuming that the multinominal cross-entropy is used for both losses, the final loss on the basis of which we train the network is given by

$$\mathcal{L} = \frac{1}{2}\left(\mathcal{L}_C + \mathcal{L}_O\right), \tag{1}$$

where $\mathcal{L}_C$ is the class loss and $\mathcal{L}_O$ is the orientation loss. During validation, we have considered the loss $\mathcal{L}$ (1) for model selection. Furthermore, the value of this function has been used to set an early stopping criterion during training (after 10 epochs without loss improvements the weight update was stopped). It is now important to define the accuracy index to be employed for performance comparison in this paper; this index is evaluated in the same way as [1]: multiple rotations (x_r, where r denoted the rotation index) of the same object are fed to the network and the final consensus on the class label is then based on the score of each rotation (S_k, where k denoted the class index), namely

$$c_{\text{final}} = \arg\max_k \sum_r S_k(x_r). \tag{2}$$

The best activation function has been selected on the basis of some tests. The authors of [1] used Leaky-ReLU. Despite this, we noticed that, if ReLU is employed, the network provides comparable results. For this reason, various tests have been performed with both activations, in order to identify the best one in each case.

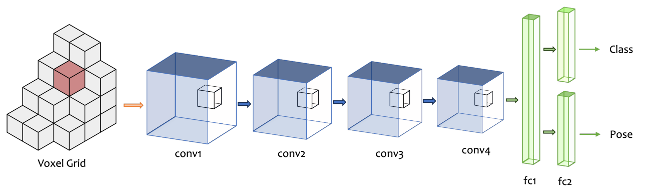


Fig. 4: ORION architecture. The voxel grid passed as input has a cubic shape.

During the implementation, we used all the original parameters of the network, especially for the dropout ratio. Batch normalisation has been also adopted.

B. RotationNet

Differently from ORION, which has a proprietary architecture, RotationNet takes its name from the idea behind its training strategy. The working environment is illustrated in Fig. 5. Its architecture is based on a pre-existing CNN followed by some classification layers, which depend on the selected basic architecture. The authors of [2] tested different CNNs as base structures like AlexNet [10], ResNet [11] and VGG16 [12]. All of them are used for 2D image analysis thanks to their ability to extract the most important features. We have implemented VGG16 only; its architecture is illustrated in Fig. 6[1].

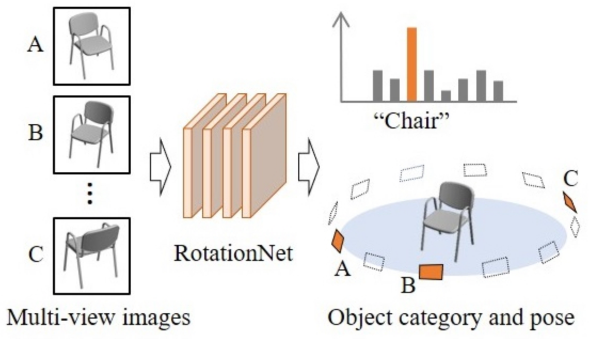


Fig. 5: RotationNet environment (figure taken from [2]).

The training process of RotationNet requires the following notions. Multi-view images of each 3D model are observed from all the predefined viewpoints. Let M and N denote the number of the predefined viewpoints and the number of target object categories, respectively. A training sample consists of M images $\{\mathbf{x}_i\}_{i=1}^M$ of an object and its category label $y \in \{1, ..., N\}$. A viewpoint variable $v_i \in \{1, ..., M\}$ is attached to each image $\mathbf{x}_i$ and set to j when the image is observed from the j-th viewpoint. Only the category label y is given during the training, whereas the viewpoint variables $\{v_i\}$ are unknown; $\{v_i\}$ are treated as latent variables to be optimised in the training process.

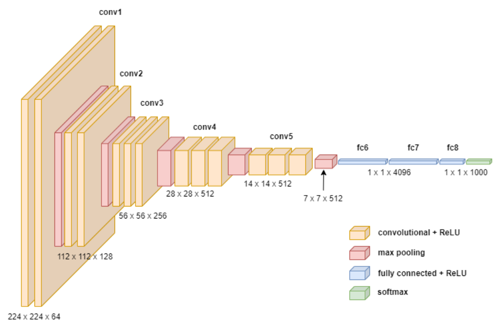


Fig. 6: VGG16 architecture.

In order to get stable solutions for $\{v_i\}_{i=1}^M$, an "incorrect view" class has been added. This requires the output of the network to be $N + 1$ dimensional. We can rewrite the optimisation problem for the network training using cross-entropy loss and some simplifications. The following details explain the procedure we adopted. To optimise the latent viewpoint, it is necessary to compute

$$\max_{\{v_i\}_{i=1}^M} \prod_{i=1}^M \frac{p_{v_i,y}^{(i)}}{p_{v_i,N+1}^{(i)}}, \tag{3}$$

where the probability $p_{j,k}^{(i)}$ is defined as

$$p_{j,k}^{(i)} = \begin{cases} 1 & (j = v_i \text{ and } k = y) \text{ or } (j \neq v_i \text{ and } k = N + 1) \\ 0 & (\text{otherwise}) \end{cases} \tag{4}$$

for any i, j and k. To obtain the last probability and subsequently simplify the evaluation of Eq. (3), the logarithmic Softmax function has been exploited; this allows us to transform the products into a summation of differences. The labels are estimated by solving the optimisation problem

$$\{\hat{y}, \{\hat{v}_i\}_{i=1}^M\} = \underset{y,\{v_i\}_{i=1}^M}{\text{argmax}} \sum_{i=1}^M \left(\log p_{v_i,y}^{(i)} - \log p_{v_i,N+1}^{(i)} \right). \tag{5}$$

It is important to stress that we did not train the entire network starting from scratch. In fact, we loaded a pre-trained version of VGG16 on ImageNet1K [13] and part of such weights, being considered to be optimal, have not been updated. In fact, the first layers of VGG16 are trained to perform feature extraction. This part does not require any improvement in our work. On the other hand, the weights of the last fully connected layers of the classifier need to be

[1]https://towardsdatascience.com/how-to-easily-draw-neural-network-architecture-diagrams-a6b6138ed875

updated. Indeed, they must be adapted to work on ModelNet10 and ModelNet40. Despite being only a few layers, due to their high dimensions, such training task is computationally heavy. Finally, it is worth mentioning that we were unable to train the whole network by ourselves due to the lack of time and of computational capabilities.

VI. Results

In this section, all the results obtained during the test of the two considered NNs are analysed in detail. First, we show some results for each of the two architectures. Then, based on these results, we provide a comparison, analysing pros and cons of eah solution. The reader can find all the useful material at https://github.com/DavidePeron19/3D_Object_Classification_NN, where a folder containg all the code and test runs is available.

A. *ORION*

In this subsection, we take into consideration the numerical results listed in Tab. 2. Based on those referring to ModelNet10, the following considerations can be made:

1) The performed data augmentation (Az6, Az12 and Az24) leads to more or less the same best and average results[2] in terms of class accuracy. Therefore, the improvements due to including orientation in NN training do not depend on the number of rotations performed.

2) In pose estimation, the fewer rotations are used, the better the accuracy, as expected. However, the performance gap is not too relevant.

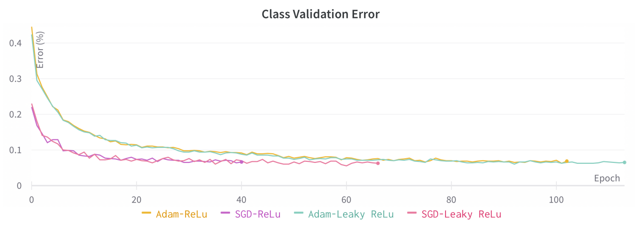
3) There is no relevant difference between the results referring to ReLu and those referring to Leaky ReLu. The activation function ReLu always gives better results, even if the difference from Leaky ReLu is approximately 0.2% on average. This result is motivated by the ORION structure. In fact, having only 6 layers, the dying ReLu phenomenon has a negligible impact.

4) Different results are obtained with distinct optimisers. In this case, tests were performed using *stochastic gradient descent* (SGD) with learning rate $lr = 10^{-3}$ and momentum 0.9, and an Adam optmiser with $lr = 10^{-5}$. If SGD is used, the obtained class accuracy is always higher. On the other hand, better pose accuracy can be achieved through Adam.

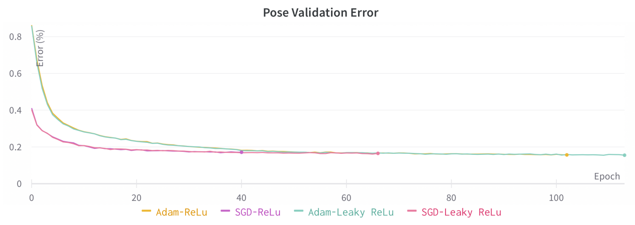
Furthermore, as it can be easily inferred from Fig. 7, the number of epochs necessary to obtain a comparable accuracy is different. The reader can easily see that SGD converges more than two times faster than Adam. This can be mainly related to SGD higher learning rate. To obtain satisfying results with Adam we needed to set a learning rate a hundred times smaller. The behaviour of the class and pose errors on the validation dataset can be observed in Fig. 7.

Let us now take into consideration the results obtained for ModelNet40. In this case the conclusions we can give were expected but, somehow, counter intuitive. As it can be easily inferred from Tab. 2, the best performance in terms

[2]Five tests for each architecture, starting every training from a different random initialisation, have been performed



(a) Class validation error



(b) Pose validation error.

Fig. 7: Behaviour of the errors for both class and pose for the validation set during the training loop w.r.t. different optimizers and activation functions.

both of class and pose accuracy is obtained using Az6 (the smallest data augmentation possible). Even if we are unable to provide a theoretical motivation for this, we can formulate the following explanation (our arguments apply to both pose and class). We strongly believe that the number of parameters of ORION is too small; consequently, this NN is unable to capture all the small features that distinguish different elements in ModelNet40. In fact, some classes differ only for some small details and this makes training more complicated; this is exemplified by Fig. 1, in which the second and third elements are very similar. One can think that they belong to the same class, but actually they belong to different categories. Furthermore, introducing rotations and approximations, this similarity effect is magnified; this makes classification harder. It is also important to keep in mind that the voxel grids used to feed the network do not have a very high resolution and they are also in their binary version. A possible solution to improve accuracy could be the use of larger voxel grid with float values. This also requires a new version of ORION with bigger layers w.r.t. the ones used in [1] and hence more parameters. As far as the performance of activation functions is concerned, we did not find any particularly relevant difference between ReLu and Leaky ReLu. Furthermore, similarly as ModelNet10, SGD performs better when optimising class accuracy, while Adam works better for pose estimation.

It is useful to make this very last remark. We were unable to exactly obtain the results shown in [1]. Despite this, if class accuracy is considered, the distance from the optimal results shown in that manuscript is negligible (less than 0.3% on ModelNet10). On the other hand, when it comes to assessing pose accuracy, things don't look quite as promising.

In conclusion, we were not able to improve upon the results illustrated in [1]. However, if our benchmarks (Az12) are taken into consideration, we have shown that using a smaller data

Data Aug.	Opt.	Act. func.	Accuracy Max (%)				Accuracy Avg. (%)			
			ModelNet10		ModelNet40		ModelNet10		ModelNet40	
			Class	Pose	Class	Pose	Class	Pose	Class	Pose
Az6	SGD	ReLu	**93.6**	85.6	**88.0**	84.9	93.4±0.07	85.3±0.11	87.6±0.18	84.1±0.27
Az6	SGD	Leaky ReLu	93.4	85.8	87.8	84.9	93.1±0.13	85.5±0.09	87.5±0.13	84.4±0.16
Az6	Adam	ReLu	93.5	**86.5**	87.5	**85.1**	92.7±0.20	86.0±0.16	87.3±0.08	84.7±0.13
Az6	Adam	Leaky ReLu	92.3	86.1	87.4	84.9	92.2±0.03	85.8±0.07	87.3±0.04	84.6±0.11
Az12	SGD	ReLu	**93.5**	84.5	**87.6**	84.2	93.4±0.04	84.0±0.17	87.2±0.16	83.5±0.19
Az12	SGD	Leaky ReLu	93.4	84.3	87.3	83.7	93.2±0.11	84.1±0.08	87.0±0.06	83.5±0.08
Az12	Adam	ReLu	92.9	**85.3**	86.7	**84.3**	92.4±0.15	84.8±0.15	86.4±0.13	84.0±0.11
Az12	Adam	Leaky ReLu	92.7	85.0	86.5	84.1	92.2±0.11	84.3±0.21	86.3±0.13	83.7±0.17
Az24	SGD	ReLu	**93.6**	83.5	**87.7**	84.2	93.4±0.06	83.0±0.21	87.4±0.11	83.9±0.09
Az24	SGD	Leaky ReLu	93.4	83.3	87.6	84.1	93.2±0.08	83.0±0.14	87.3±0.10	83.8±0.10
Az24	Adam	ReLu	92.7	**84.1**	87.3	**84.5**	92.4±0.09	83.7±0.11	87.0±0.08	84.1±0.11
Az24	Adam	Leaky ReLu	92.6	83.9	87.2	84.4	92.3±0.11	83.6±0.11	86.9±0.11	84.0±0.18

TABLE 2: Training results for ORION. The results listed in this table have been obtained by performing 5 different training runs for each combination of optimiser, activation function and data augmentation. The best results in terms of pose and class for each data augmentation are evidenced in bold, whereas the overall best results are evidenced in red.

augmentation, namely Az6, comparable or even better results can be obtained while also reducing training complexity. Such improvements can be observed on ModelNet datasets, but we do not know how they generalise in other scenarios.

B. RotationNet

Let us now make some considerations about RotationNet, that provides some really interesting results, listed in Tab. 3. In case (i), it is possible to state that the obtained performance does not look so impressive, given the complexity of the employed NN. However, in this case, the NN has been trained using only on a subset of the full dataset. Such a small size, especially for ModelNet10, makes the training of the classifier harder and worsens its ability to generalise on the test set. In case (ii), outstanding results in terms of accuracy are obtained. This is due to both the network architecture and the way it is trained. Indeed, RotationNet, being a deeper network, is able to capture more image features. Furthermore, the use of 20 different viewpoints of each image allows the NN to achieve a better comprehension of each model.

It is important to remark that RotationNet requires a huge amount of time to be trained. For this reason, we were forced to make a single training run for each case and dataset, also with a limited number of epochs. Given the hardware at our disposal, the training time on ModelNet40 (case (ii)) for about 40 epochs was approximately 3 days. This fact also explains why our results are a bit worse that the ones that can be found in [2].

The trend of the validation error and training loss during the 4 different training runs is shown in Fig. 8 and 9, respectively. As it can be easily inferred from this figures, the error is still decreasing when training stops. This leads that the conclusion that the model can still improve, but this requires too much time.

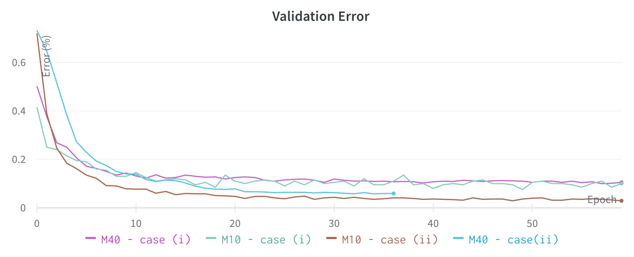


Fig. 8: Validation error in different training cases; both ModelNet10 (M10) and ModelNet40 (M40) are considered.

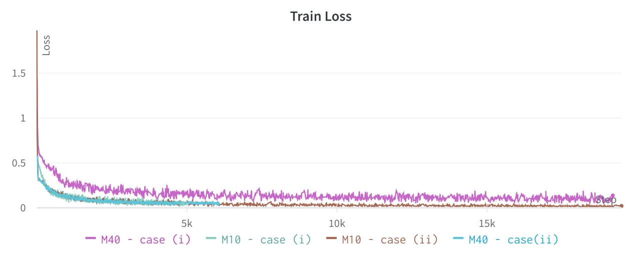


Fig. 9: Training loss behaviour; both ModelNet10 (M10) and ModelNet40 (M40) are considered.

C. Comparison

In order to compare the performance achieved by ORION and RotationNet, we can start from Tab. 3. The results listed in that table evidence that the use of multiple orientations during training influences the final accuracy.

In case (i), ORION, despite its simpler architecture, provides comparable or worse results. In our opinion, this is due to the used training data. In fact, ORION is trained using 12 3D models for each object, while RotationNet exploits the same number of samples but in 2D. Hence, the amount of information that can be captured is smaller. Such poorness of data is relevant especially for ModelNet10. On the other hand, things are slightly better for ModelNet40. In fact, since this set contains a lot of very similar classes, differences are better captured by 2D images.

In case (ii), instead, RotationNet substantially outperforms ORION for both ModelNet datasets. In fact, the achieved accuracy is at least 3% higher. This result is due to the higher number of viewpoints used during training. This abundance of information allows RotationNet to capture also some small details that otherwise would be lost. Such a difference can be observed especially when referring to ModelNet40, where the improvement in class accuracy is greater than 5.5%.

In conclusion, RotationNet outperforms ORION. This can be justified by the significant differences in their architectures. ORION has slightly more than 4M parameters, while RotationNet has more than 100M parameters. This allows to obtain a better feature detection.

Method	Data Aug.	Accuracy (%)	
		ModelNet10	ModelNet40
ORION	Az6	93.6	88.0
ORION	Az12	93.5	87.6
ORION	Az24	93.6	87.7
RotationNet	Case (i)	92.5	90.0
RotationNet	Case (ii)	**96.8**	**93.7**

TABLE 3: Accuracy achieved by the two different NNs we tested. Only the best result for each set or rotation is provided.

VII. Concluding Remarks

Our work has mainly focused on comparing different CNN architectures for 3D object classification. Due to limited time and resources, we were unable to obtain exactly the results of the manuscripts we took as reference (namely, [2] and [1]). Despite that, the efforts required to write the necessary code and analysing the results has allowed us to fully understand some crucial aspects of neural network design and training.

Based on our study of some technical papers and the results we obtained, the following considerations can be made:

1) The employed dataset and the way it is used play fundamental roles. In fact, if the dataset does not collect enough samples, overfitting may represent a serious problem.

2) In some cases, the authors of the technical papers considered in our work did not explicitly explain how their validation set was generated and, in particular, they used the test set as a validation set. In such a scenario, especially when the dataset was small, we made the same choice. However, for larger datasets (obtained through data augmentation), we preferred to create a validation dataset. As shown in Section IV, depending on the validation dataset exploited for model selection, a given NN performs differently on the test set. All this may explain the differences between some results we obtained and those shown in the above-mentioned technical papers.

3) In network training, substantial attention must be paid to the selection of learning rate, since convergence should be fast and good at the same time. During our tests, it has emerged that adopting a higher learning rate during the first iterations makes the training phase significantly faster; however, we should avoid overshooting the minimum.

4) Initialisation is also crucial. In fact, starting from a good initial point leads, for a given batch size, to a shorter run time. In fact, such condition granted a shorter training time for each epoch and at the same time better accuracy overall.

5) The computational power required to train a NN can be really significant. During our tests, we used a fairly powerful GPU for home use, namely an NVIDIA 1080Ti with 3584 Cuda Cores. We were not able to use Google Colab due to storage restrictions, that did not allow us to store the desired datasets with data augmentations. Our tests on ORION required an acceptable computing time. This is because we were training a network made only of a few layers of small size with a relatively small number of parameters. When we started working on RotationNet, the required time grew exponentially. This is due to the fact that the memory necessary to store every batch was much larger. Furthermore, since VGG16 contains more than 100M parameters (instead of less than 5M is the case of ORION), the time required to complete a single epoch was longer. This explains the importance of using a dedicated hardware able to parallelize training operations.

VIII. Individual Contributions

When evaluating the individual contribution provided by each member of a given team, it can be challenging to describe the precise extent of the role played by every person. We chose to collaborate closely throughout the project, often working together on a single machine, especially when dealing with the time-intensive data preprocessing required for ORION and the subsequent training. Despite this, we can state that Emanuele Vitetta primarily focused on the issue of data preprocessing, whereas Davide Peron concentrated on the training loops. It is worth pointing out that the two members of the team jointly designed the neural network architectures. Furthermore, each of them has reviewed and corrected the work done by the other one. Finally, as far as this manuscript is concerned, it has been jointly written in the Overleaf collaborative environment; similar contributions have been provided by the co-authors.

References

[1] N. Sedaghat, M. Zolfaghari, E. Amiri, and T. Brox, "Orientation-boosted voxel nets for 3d object recognition," in *British Machine Vision Conference (BMVC)*, 2017. [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2017/SZB17a

[2] A. Kanezaki, Y. Matsushita, and Y. Nishida, "Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints," in *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[3] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," 2015.

[4] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *Ieee/rsj International Conference on Intelligent Robots and Systems*, 2015, pp. 922–928.

[5] A. Kanezaki, Y. Matsushita, and Y. Nishida, "Rotationnet for joint object categorization and unsupervised pose estimation from multi-view images," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 43, no. 1, pp. 269–283, 2021.

[6] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "Modelnet dataset," 2015. [Online]. Available: https://modelnet.cs.princeton.edu/

[7] A. Kanezaki, Y. Matsushita, and Y. Nishida, "Modelnet png dataset," 2018. [Online]. Available: https://github.com/kanezaki/rotationnet

[8] N. Sedaghat and T. Brox, "Unsupervised generation of a viewpoint annotated car dataset from videos," in *IEEE International Conference on Computer Vision (ICCV)*, 2015. [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2015/SB15

[9] N. Sedaghat, M. Zolfaghari, E. Amiri, and T. Brox, "Orion original implementation," 2017. [Online]. Available: https://github.com/lmb-freiburg/orion

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.