AURORA

AUgmented Recognition ORientation-boosted Architecture

Giovanni Girardin , Alberto Morselli , Riccardo Rettore
Department of Information Engineering, University of Padova

***Abstract*—Recent advances in 3D object recognition have leveraged convolutional neural networks (CNNs) to improve classification performance. This work investigates an enhanced network architecture, strongly inspired by the ORION model, which augments traditional voxel-based 3D classification by incorporating object orientation as an auxiliary task. By framing orientation estimation as a classification problem, the network learns a more robust feature representation, enhancing the accuracy of its primary object classification tasks. The experiments were carried out on two different datasets, ModelNet10 and ModelNet40, both modified ad hoc. This research highlights the importance of multitask learning in 3D recognition and provides insights into effectively applying orientation enhancement to improve deep learning models for real-world 3D data.**

***Index Terms*—3D Object Recognition, Convolutional Neural Networks (CNNs), Voxel Grids, 3D Object Orientation, ModelNet40.**

I. Introduction

3D object recognition is a key challenge in the domain of computer vision, with wide applications in robotics, autonomous vehicles, and augmented reality. Recent advances in deep learning and CNNs have made significant strides in improving the performance of this task. Despite these advances, many existing methods still struggle to achieve high accuracy when dealing with objects in varying orientations.

This paper introduces a novel approach that enhances 3D object recognition by incorporating orientation estimation as an auxiliary task alongside the main classification task. By learning to predict not only the class label of an object but also its orientation, the network becomes more robust to changes in the object's pose, leading to improved classification accuracy.

To achieve this, we build upon the well-known ModelNet dataset, applying data augmentation to create a more diverse set of training samples, and labeling of the original dataset in order to be able to add the secondary task.

The architecture proposed in this paper follows the VoxelNet framework [1]. The network is trained on voxelized 3D objects. We evaluate our method on the ModelNet10 and ModelNet40 datasets, with two slightly different architectures and different orientation weight hyperparameters to find the best results.

II. Paper Structure

Section III provides an overview of the related work on the subject, which served as an inspiration for the development of the Learning framework discussed in Section IV. The latter is further elaborated in terms of Signals and Features in Section V, its Architecture in Section VI, and the Loss topic is treated in Section VII. The Method behind our workflow is discussed in Section VIII, concluding with the presentation of our Results in Section IX, from which we derive the Concluding remarks in Section X.

III. Related work

The starting point for this work is the Orientation-Boosted Voxel Nets (ORION) model proposed by Sedaghat et al. [1], which enhances 3D object recognition by incorporating orientation estimation as an auxiliary task. ORION builds upon earlier voxel-based methods, such as VoxNet by Maturana and Scherer [2], a novel approach that introduced a 3D convolutional neural network (CNN) architecture for recognizing voxelized 3D meshes. Our work builds upon the ORION framework, further refining it by introducing custom orientation binning strategies and enhanced data preprocessing techniques. These improvements aim to address the limitations of previous methods, particularly when dealing with highly symmetric classes and random orientations. We call our network AURORA: AUgmentede Recognition ORientation-boosted Architecture.

IV. Learning Framework

The proposed learning framework is designed to simultaneously address the tasks of 3D object classification and orientation estimation through a multitask learning approach. The framework leverages a modified 3D CNN architecture that processes voxel grid inputs and outputs predictions for both class and orientation labels.

The architecture is based on 4 convolutional layers followed by two fully connected ones. The pooling and dropout layers are incorporated, as well as the filter stride to progressively reduce the input size. The final layers branch into two parallel heads: one for class prediction and one for orientation prediction.

The network employs a multi-task loss, convex combination of two multinomial cross-entropy losses:

$$\mathcal{L} = (1-\gamma)\mathcal{L}_{\mathcal{C}} + \gamma\mathcal{L}_{\mathcal{O}} \tag{1}$$

where $\mathcal{L}_{\mathcal{C}}$ is the classification loss, $\mathcal{L}_{\mathcal{O}}$ is the orientation loss, and $\gamma \in [0, 1]$ is a weighting parameter that balances the two tasks. This design ensures that both tasks are learned jointly, with the orientation task acting as an auxiliary task to improve feature representation for the primary classification task.

The network is trained using an augmented and offline pre-processed ModelNet dataset, with voxel grids of fixed size as input. A manual labeling work carried out on a data-augmented dataset has been necessary in order to add the orientation auxiliary task, both prior to training.

To address the challenge of different symmetry between classes we assign a different bins size (leading to a different amount of output neuron) to each class. This adjustment ensures that the orientation loss remains meaningful and improves training stability.

Dropout layers and batch normalization techniques ensure that the network effectively learns robust features for both classification and orientation estimation.

V. Signals and Features

A. Original Dataset

The original ModelNet dataset provides 3D object meshes stored as .off (Object File Format) files, which represents the geometry of an object as a list of vertices and the faces that connect them. This simple format is widely used for 3D modeling due to its ease of parsing and flexibility. Despite the difference in the number of classes, both versions of it (ModelNet10 and ModelNet40 [3]) share the same underlying structure. Each class comes in his dedicated folder, pre-divided in training and testing data. Smaller classes, such as xbox, bathtub, and door, include slightly over 100 training samples, while larger classes, such as airplane and bookshelf, have almost 700 samples.

It's also worth mentioning that the dataset is uniformly aligned, simplifying preprocessing and ensuring a consistent basis for data augmentation techniques, as done in our work.

B. Pre-processing

A first step is to transform raw 3D object meshes into voxel grids, a representation that partitions space into uniform cubic units called voxels. Preprocessing can be handled in two ways:

- **Offline**: Preprocessing the dataset prior to training mitigates memory allocation issues and speeds up training. However, adding data augmentation requires reprocessing the entire dataset.
- **Online**: This method offers flexibility but introduces significant time costs, especially when training with different architectures or hyperparameters.

This process standardizes the objects into $32 \times 32 \times 32$ voxel grids, which serve as input to the neural network. This step also reduces the size of the meshes, rescaling them to a consistent dimension at the cost of some finer details. The results of this process are shown in Fig. 1.

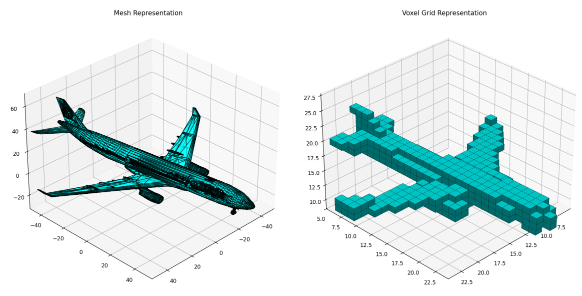


Fig. 1: Mesh voxelization process

C. Data augmentation

To enhance orientation awareness, each mesh is rotated along the z-axis in 20° increments, resulting in 18 augmented samples per mesh spanning 0° to 360°. In addition, small random noise ($\pm 2°$) is added to improve the robustness to slight orientation variations. An example is shown in Fig. 2.

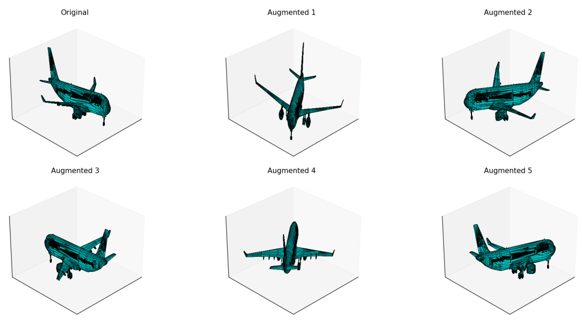


Fig. 2: Offline data augmentation

D. Manual labeling

After augmenting the dataset, it is also necessary to address the lack of orientation labels for each mesh. To address this, each 3D object is assigned to an orientation value represented by a discrete number ranging from 0 to 17. These labels correspond to orientation angles spanning 0° to 360° in 20-degree increments, ensuring consistent and structured annotations across all meshes (e.g., any rotation between [0° and 20°) corresponds to label 0, any between [20° and 40°) to label 1 and so on). For convenience, the label was appended to the end of each file name, making it easy to access and integrate into the code.

E. Compression

Training directly using the voxel data in the .npy format was found to be extremely slow and inefficient. To ensure efficient data handling and quick access during training, the voxel grids were then compressed and stored in a HDF5 file format. This way of handling the data allows the network to speed up read and write operations, improving the overall training time, according to our experiments, by up to almost 10 times, highlighting the significant performance benefits of using HDF5 for large-scale datasets. The whole pipeline for dataset processing is pictured in Fig. 3.

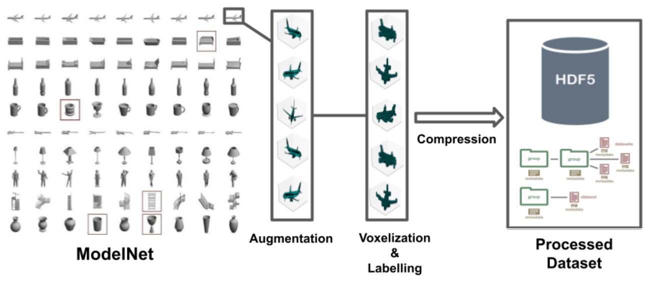


Fig. 3: Offline processing

VI. Architecture

The baseline architecture shown in Fig. 4 is also based on [2] and subsequently modified as recommended by [1]. This architecture takes a 3D voxel grid as input and processes if with four 3D convolutional layers, followed by two dense layers. Following [1], we further refine it by incorporating orientation estimation as an auxiliary parallel task.

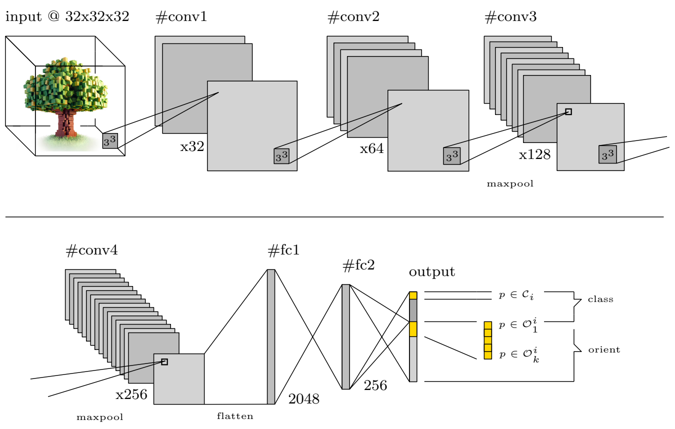


Fig. 4: Model architecture

Without loss of generality, attention is given only to rotation about the z-axis, identified as the most significant component in practical scenarios. In the following, the 'orientation' term is used to refer only to this component. Although orientation is inherently a continuous variable that the network could predict, treating this as a regression problem, the approach employed treats various orientations of an object distinctly, thus recasting orientation estimation as a classification problem. The network is designed with output nodes corresponding to the product label space of classes and orientations, thereby learning the mapping $x_i \to (c_i, o_i)$ where x_i denote the input voxel grid samples, and c_i, o_i represent their respective object and orientation classes.

A. Body

The architecture proposed by [1] has been followed pretty closely, with the exception on the final layers. We adopt a gradually increasing dropout probability as we go deeper across the network layers to achieve enhanced regularization in the last ones, which are typically where higher-level, abstract features emerge. This incentivize the model to depend on more distributed representations, particularly in the fully connected layers close to the output. We also make use of leaky ReLU to prevent the "dying ReLU" problem.

To prevent excessive dimensionality that may lead to overfitting, we incrementally reduce the sample dimensions making use of stride and pooling layers as we progress toward the input of the fully connected layers. This approach maintains a manageable feature representation size, optimizing the model's ability to generalize effectively.

Layer	Shape	Parameters	BN
conv1	x32	k=3, s=2, p=1	✓
leakyReLU	-	a=0.1	-
drop1	-	p=0.2[1]	-
conv2	x64	k=3, s=1, p=1	✓
leakyReLU	-	a=0.1	-
drop2	-	p=0.3	-
conv3	x128	k=3, s=1, p=0	✓
leakyReLU	-	a=0.1	-
pool3	-	MAX, k=2, s=2	-
drop3	-	p=0.4	-
conv4	x256	k=3, s=1, p=0	✓
leakyReLU	-	a=0.1	-
pool4	-	MAX, k=2, s=2	-
drop4	-	p=0.6	-
flatten	256x2^3 $\to$ 2048	-	-
fc1	2048 $\to$ 256	-	-
leakyReLU	-	a=0.1	-
fc2	256 $\to$ variable[2]	-	-

TABLE 1: Detailed architecture.

B. Output Layer

Attached to the end of #fc2 we connect the output layer, whose structure deserve additional care. Due to the diverse symmetries between the different objects, the orientation task is addressed differently across different classes, as also suggested by [1]. The intuition behind it is straightforward: a flower pot exhibits significantly greater symmetry compared to a table, such that even a human observer might find it challenging to distinguish it from its 180° rotated version. Consequently, it is reasonable to question why a neural network model would be expected to make such a distinction.

Adopting this perspective implies that objects with greater symmetry deserve a relaxed approach regarding the precision of orientation values, also kept in account in the validation metrics discussed in Section VII-B. Choosing to address the orientation task as a classification problem, as opposed to a regression one, simplifies the process. This translates into using quantized angle bins and solving the problem by determining the specific bin into which the object fall.

Following the idea entails assigning a greater number of quantized bins (output neurons) to classes with lower symmetry, while fewer to those with higher symmetry, till the extreme case of rotational symmetry where we assign only

[1]where k = kernel size, s = stride, p = padding/drop probability, a = slope.
[2]ModelNet10: 10+44, ModelNet40: 40+190 (see Section VI-B)

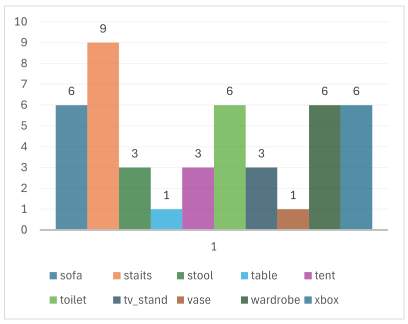
a single output neuron, thereby effectively disregarding the orientation task for that specific class. The choice on the number of neurons in the output layer corresponds then to

$$\text{output neurons} = k + \sum_{n=0}^{k-1} o_n \tag{2}$$

where k is the number of classes and o_n is the number of neurons reserved to each class. The choice of them is based on a sistematic method explained in the next section.

C. Orientation neurons

The number of orientation neurons is inferred by a class asymmetry score obtained rotating a significant amount of samples for each class by 90° and 180°, performing the IoU metric and then averaging over the class. Asymmetric classes gets a higher score, while symmetric a lower one. We then normalize the score by a $\tanh(\cdot)$ like function to stretch the diversity score, and, based on the asymmetry score, round it to 1x, 3x, 6x, or 9x bins of 360°, 120°, 60° or 40°. The values for ModelNet10 and ModelNet40 are shown respectively in Fig. 5 and Tab. 2.



Class	#
sofa	6
stairs	9
stool	3
table	1
tent	3
toilet	6
tv_stand	3
vase	1
wardrobe	6
xbox	6

Fig. 5: Custom ModelNet10 Orientation neurons for each class

Class	#	Class	#	Class	#
airplane	6	flower_pot	3	radio	3
bathtub	6	glass_box	3	range_hood	6
bed	1	guitar	9	sink	6
bench	6	keyboard	1	sofa	6
bookshelf	6	lamp	3	stairs	9
bottle	1	laptop	6	stool	3
bowl	1	mantel	6	table	1
car	3	monitor	9	tent	3
chair	6	night_stand	3	toilet	6
cone	1	person	6	tv_stand	3
cup	3	piano	9	vase	1
curtain	9	plant	6	wardrobe	6
desk	6	dresser	6	xbox	6
door	6				

TABLE 2: ModelNet40 Orientation neurons for each class

VII. Loss

We choose multinomial cross-entropy loss that takes into account both classification and orientation, as

$$\mathcal{L} = (1-\gamma)\mathcal{L}_{\mathcal{C}} + \gamma\mathcal{L}_{\mathcal{O}} \tag{3}$$

where $\mathcal{L}_{\mathcal{C}}$ and $\mathcal{L}_{\mathcal{O}}$ indicate respectively losses for classification and orientation tasks. Since the classification training is intrinsically boosted by orientation training strengthening γ, even if it seems to indicate giving less importance to our primary classification task, it can implicitly lead to improvements. In our experiments, it showed that this can be used as epochs advance to stir the gradient if stuck somewhere.

Due to the various object symmetries and our choice to handle this as explained in section Section VI-B also the loss has to be adapted to this new scheme, with the considerations discussed in the next section.

A. Label rescaling

As introduced in Section V-D true labels $\in [0, 17]$ (where label 0 represent an orientation value $\in [0°, 20°)$, label 1 a value $\in [20°, 40°)$ and so on…) gets rescaled to correspond to the output neurons they should point to, using the formula:

$$y'_{O_i} = N_i(y_{C_i}) + y_{O_i} \ // \ [18//n_i(y_{C_i})] \tag{4}$$

where y_{C_i} is used to get the starting point of the group of neurons belonging to that class, indicated by N_i, while y_{O_i} is used to compute the offset rescaled with respect to the number of neurons n_i assigned to that class.

For example (referring to Tab. 2) a bathtub sample (y_{C_i}=1) with orientation label y_{O_i}=2 becomes $y'_{O_i} = 6+2//(18/6) = 6+2//3 = 6+0 = 6$, since with 6 neurons assigned to its class, every orientation label between $[0-3]$ falls in the first bin.

B. Validation Accuracy

Even not our primary task, tracking the orientation accuracy can provide significant insights into the training behavior of our network. Due to the high-sparsity structure of our orientation classification task (predicting 1 label out of 190) the conventional classification metric (assigning 1 if correct and 0 otherwise) tends to be uninformative, as it would consistently yield results almost around zero regardless of the network's training status.

An initial approach to addressing this problem involves using a top-3 or top-5 metric that would help classes with a large amount of neurons but would be excessively generous with classes with merely 3, or, even worse, only a single neuron, leading to an unrealistic measure of goodness that does not reflect the actual accuracy of the model.

We finally adopted a flexible approach that considers the number of neurons that each class has, thereby selecting between a top-3, -2, or -1 (the default) metric based on whether the class contains 9, 6, or (3/1) neurons, respectively, to ensure balanceness.

VIII. Method

During the development of this project, several approaches were explored but ultimately discarded due to inefficiencies or limitations. These attempts, while not included in the final pipeline, provided valuable insights and highlighted potential areas for future exploration, and we gave them a mention here:

- **Dataset in .npy format:**
  The frequent read and write operations in .npy format caused unnecessary delays, making it impractical for large-scale experiments and also for training multiple architectures while looking for the best structure.
- **18-neurons orientation output layer:**
  Another discarded attempt involved using a single output neuron per orientation bin (totaling 18 neurons) for classification, without considering the object class. While this approach simplified the output structure, it introduced significant challenges for objects with radial symmetry (e.g., vases or tables), where distinguishing between orientations was inherently ambiguous. This led to poor orientation loss calculations, ultimately reducing the accuracy of the model.
- **Voxel grid size variations:**
  While the final implementation used $32 \times 32 \times 32$ voxel grids, other sizes were considered during experimentation. An initial approach used $33 \times 33 \times 33$ grids with similar results, used to make the combination between filter's stride and kernel size perfectly cover the input without asymmetric padding. It has been later changed to a $32 \times 32 \times 32$ size due to a power-of-2 dimensions reason, which is better optimized for certain hardware architectures and showed an improvement in training speed.

IX. Training and results

We trained both the architectures trying multiple batch-sizes and γ valued, concluding with similar results where the difference in terms of accuracy is not significantly big to allow us to state something with enough confidence. Altought, we noticed a difference in terms of training speed, as already mentioned in the previus section, and so it's worth mentioning it. We show here the results of the original work we took inspiration from [1], and our network trained with a fixed batch size = 64, and several γ values after 5 epochs.[3]

Model	γ	**Orientation (%)**	**Classification (%)**
ORION		87.6	89.7
AURORA	0.0	-	82.3
	0.3	64.3	86.2
	0.5	64.4	87.2
	0.7	63.2	86.5

TABLE 3: Accuracy results of the different models

[3]AURORA with $\gamma = 0.5$ trained over 10 epochs performed 87.6 for classification and 65.8 for orientation.

After finalizing the hyperparameter selection, we proceeded with training the model for a total of 10 epochs. This number was determined based on the analysis of the validation curves, which indicated convergence of the model's performance metrics and stabilization of the loss function by this point. Training beyond 10 epochs did not provide any significant improvement in validation accuracy.

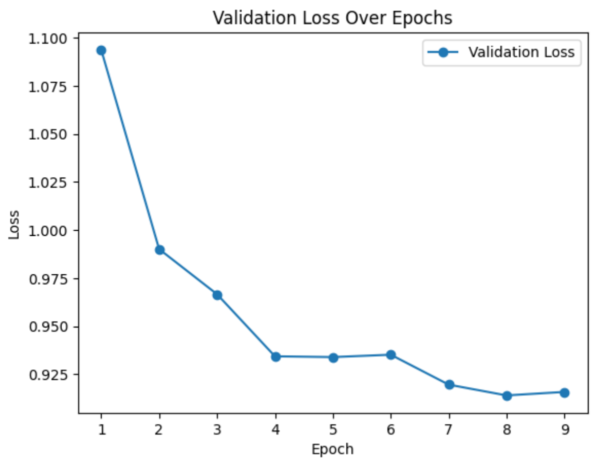
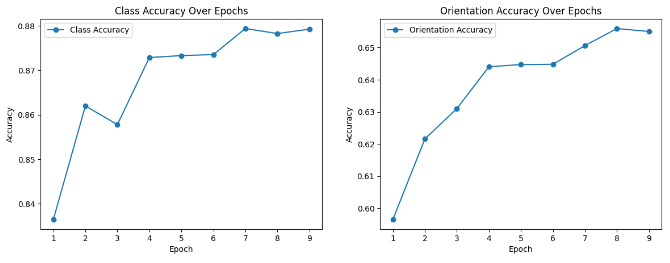


Fig. 6: Validation loss

Tracking the model's performance across epochs provides valuable insights, particularly because the network is simultaneously learning to classify both the orientation and the class of each 3D mesh. Plotting the validation accuracies for both tasks helps assess the model's ability to balance these objectives.

As shown in the plots below, both classification and orientation accuracies exhibit a positive trend as the loss decreases, indicating that the model is successfully learning both tasks. This also demonstrates that the model is not overly focusing on the primary classification task at the expense of orientation estimation, but rather improving in both metrics simultaneously.



(a) Classification (b) Orientation

Fig. 7: Accuracy over epochs of the final model with $\gamma = 0.5$.

As additional measure of performance, in Fig. 8 is shown the confusion matrix at the final validation step between all 40 classes.

Notably, the only off-diagonal value exceeding 0.3 is between classes 15 and 26, corresponding to 'flower pot' and

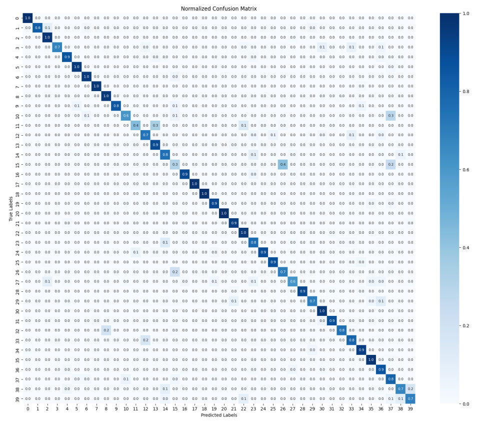


Fig. 8: Confusion matrix

'plant', which is intuitively understandable given their similarity. Finally, the model is evaluated using a set of randomly oriented meshes. The orientation labels for these meshes are assigned based on the closest bin to their randomly generated orientation. For example, a mesh with a random rotation of 17° is assigned to the 20° rotation bin and labeled as 1.

Dataset	**Orientation(%)**	**Classification(%)**
Random oriented	36.25	81.9

TABLE 4: Accuracy results for random oriented dataset

Examining the results, we observe a significant decrease in performance when the model is tested on a randomly oriented set of meshes. Specifically, the model correctly predicts the orientation approximately one-third of the time, which may not be sufficient to substantially enhance classification accuracy.

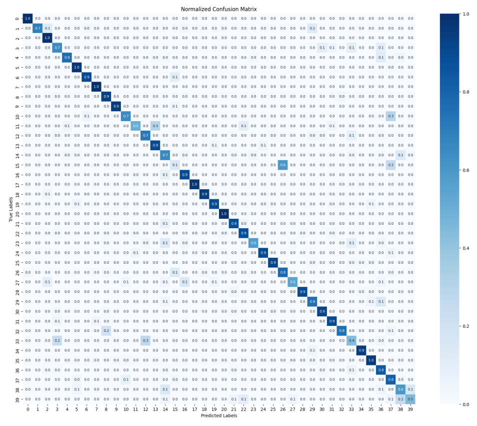


Fig. 9: Confusion matrix on random orientated meshes

A closer analysis of the confusion matrix in Fig. 9 reveals a noticeable trend: classes with high accuracy on the original dataset maintain strong performance, whereas classes that initially performed poorly experience a pronounced decline. Notably, 'plant' and 'flower pot' are effectively treated as a single class by the model, which struggles to differentiate between them. A similar pattern is emerging for other class pairs, such as 'desk' and 'table', further highlighting the model's limitations in handling ambiguous or visually similar categories under random orientation conditions.

X. Concluding Remarks

A. Considerations and future development

The results demonstrate that incorporating orientation estimation into the model is a promising direction for enhancing classification accuracy. However, when tested on randomly oriented meshes, the model's performance declined notably compared to the standard case. This suggests that the current orientation bin size of 20° may be too coarse for certain classes, and reducing the bin size could further improve the accuracy. Future work will aim to extend the model to handle rotations along all three axes. This can be achieved through data augmentation to include all possible poses of 3D objects. Although straightforward, this approach presents a significant challenge, the resulting increase in dataset size may dramatically escalate computational demands, necessitating more efficient data handling strategies.

B. Learning outcomes and challenges

When approaching this problem for the first time, several key challenges were encountered, which required both technical solutions and optimizations. The main difficulties included:

- The ModelNet40 dataset contains meshes of varying dimensions that require voxelization before training. While voxelization is straightforward, it becomes computationally expensive when combined with multiple rotations for data augmentation. Two solutions were explored:
  1) Rotating voxelized representations instead of raw meshes, which reduced computational cost by voxelizing each mesh only once. However, this approach introduced errors due to the coarse voxel resolution.
  2) Offline pre-processing of voxelization and augmentation, which was ultimately implemented. This solution stored pre-processed data locally, enabling efficient training and hyperparameter testing with a ready-to-use dataset.
- Once offline pre-processing was in place, a new challenge emerged: efficient data retrieval during training. Initially, the dataset was stored as boolean arrays, but this structure proved inefficient for the dataloader, causing slowdowns. After testing various formats, the HDF5 file format was adopted, resulting in a 10-fold improvement in data loading speed and enabling smoother, faster training iterations.

References

[1] N. Sedaghat, M. Zolfaghari, and T. Brox, "Orientation-boosted voxel nets for 3d object recognition," *CoRR*, vol. abs/1604.03351, 2016.

[2] 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)*, pp. 922–928, 2015.

[3] "Original modelnet 10/40 dataset." https://3dshapenets.cs.princeton.edu/.