Ensembling CNN models for 3D Object Classification based on Voxel Grid representation

Simone D'Antimo†, Harjot Singh†, Jacopo Pegoraro†

***Abstract*—Convolutional Neural Networks (CNN) have been widely used as Deep learning tools to enhance the spatial relationship among images and videos. In this paper, we illustrate the potential of CNN and Voxel grids for the 3D Object classification task. Intuitively, given a 3D image, we split it into blocks, each of those blocks may be only full or empty, which implies that the resolution of the 3D object is strongly correlated with the number of blocks in which they are split in, as trade-off more blocks implies more memory required for storage. We decided to evaluate the potentiality and the limits of a 3D Object Classification System having a limited amount of computational capacity and memory storage: those limitations imply that we must use samples with a lower resolution (i.e. composed by a limited number of voxel), loosing some details and shape of the objects, but at the same time the results obtained show that we created a robust model which perform efficient analysis and can classify the objects independently of their rotation. We were able to reach good results on the test set, with an accuracy rate of 88 percent at the end.**

***Index Terms*—Neural Networks, Convolutional Neural Networks, 3D Object Classification, Voxel Grid, ModelNet10, Data Augmentation.**

I. Introduction

The 3D Object recognition task contributes to the field of computer vision by enabling the detection, classification and also localization of three-dimensional objects in images or video streams. Thanks to the improvements in Computer Vision and Deep Learning technologies these tasks are more and more reliable and accessible, enabling many emerging applications like augmented reality, virtual reality and many more related to robotics interactions with the environment. The way 3D data is represented has a significant impact on the performance of the 3D object detection. Here we introduce two of the most common representation for 3D Objects.

- The Point-based representation uses a set of 3D points to describe the geometry of an object. This representation is lightweight and can handle large point clouds efficiently. The structure can retain precise point positions, but at the same time it has a limited feature representation and since the points are unordered using them requires a high computational capacity [1].
- Voxel-based representation models the object as a 3D grid of voxels, where each voxel has a binary value to indicate whether the object occupies it or not. This representation provides detailed information about the shape and structure of objects, but it requires a lot of memory, since we have to store also the empty blocks, therefore they are not scalable as other approaches [1].

In this paper, we propose a "label predictor" that relies on 5 models using an ensembling technique to classify 3D objects, using at the same time a coarse voxel granularity. As mentioned above, the Voxel Grid structure maintains information regarding the shape and structure of the 3D Object, in order to properly capture and maximize the spatial relationship provided by the Voxel-Grid structure, we decided to use CNNs as core networks.

The main advantages of using this kind of approach is that each weak model is trained on a portion of the dataset, requiring smaller and thus quicker models. Since the weak models are trained on different portion of the dataset, they learn and make prediction based on different features, increasing the generalization of our ensemble system. If one can run all 5 weak models in parallel, and then choose the final label, this may result faster than running a single bigger model.

The memory requirements for voxelized 3D objects increase cubically with the resolution. We have seen that, for a resolution of 16, where each object is divided in a 16x16x16 voxel grid, the visualization is too coarse to be used. On the other hand, a resolution of 64 requires 262.400 blocks for each object, which is too much considering our memory capacity. In the end, we went with the resolution of 32x32x32, which has an affordable amount of blocks per objects ($\sim$32.000) [1]. In our case, this resolution is the optimal trade-off between memory and information loss. The expectation is that using a restrained number of blocks for each Object, the model will be more efficient by taking less time to execute and requiring less memory to storage all the Voxels information.

The dataset which we used for training our models is a modified version of the ModelNet10 [2]. The initial dataset is composed by 4899 aligned 3D CAD model divided in 10 different classes, we perform data augmentation to create a portion of rotated entries, which is useful to make our models learn how to classify objects independently of their rotation in the Z axis.

The rest of the paper is organized as follows: Section II provides an overview of related works; Section III describes our approach; In Section IV we describe in details the dataset and pre-processing techniques; In Section V we present the Convolutional Neural Network that we built, summarizing the parameters we choose; In Section VI we discussed our main findings; finally, Section VII draws some conclusions and outlooks for future works.

†Department of Information Engineering, University of Padova,
email: {simone.dantimo, harjot.singh}@studenti.unipd.it,
jacopo.pegoraro@unipd.it

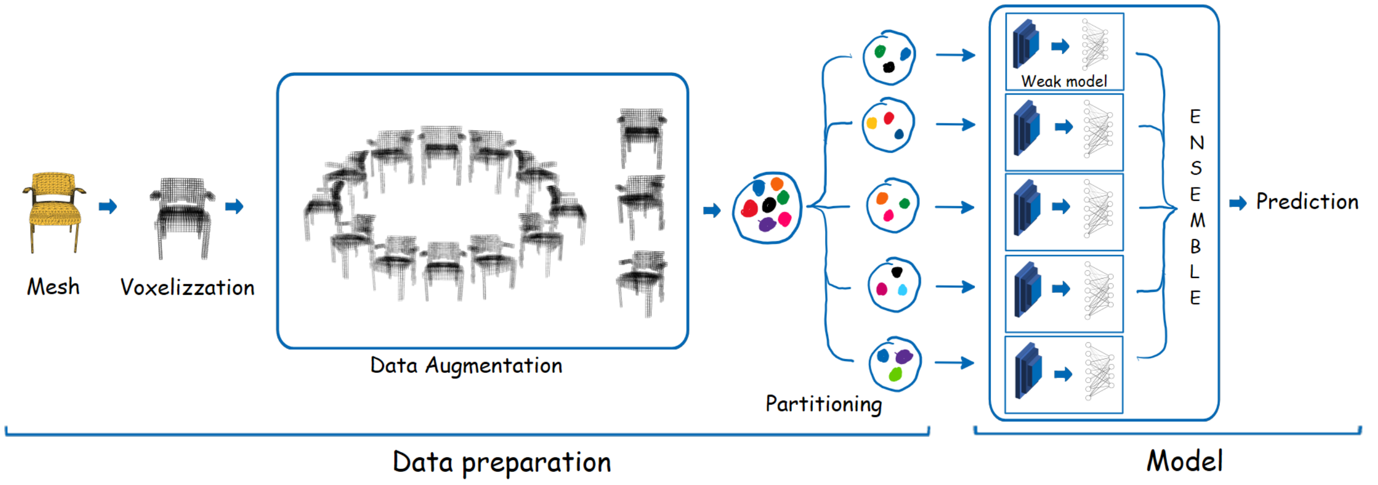


Fig. 1: Processing Pipeline. Starting from left, we have the voxelization of the ModelNet10 dataset, followed by data augmentation with new rotations. Next, we have the partitioning of the rotated dataset into five different subsets. The subsets are give as input to the bagging models for the training.

II. Related Work

In recent times, object recognition has gained significant attention with the advancements in deep learning and computer vision technologies. These developments led to remarkable progress in the ability of computer systems to identify and recognize objects within images and videos. Many 3D Objects datasets are becoming free and accessible; Different deep learning models have been trained and tested on vast amounts of data and have exhibited good learning and generalization capabilities, surpassing the performance of traditional computer vision approaches. The generalization of the learned feature implies more capability to recognize an object even if it is rotated or is seen with a different angle, a paper that talks about this and create a model that may recognize an object regarding its orientation is ”Orientation-boosted Voxel Nets for 3D Object Recognition” [3], they used pre-trained Convolutional Neural network from the VoxNet [4] and re-trained to classify in addition also the rotations of the sample, this showed a clear improvement in the overall accuracy, going from 92% to 93.9 on the ModelNet10 dataset.

In the corresponding field of 3D Object Detection, to overcome one of the challenges that may arise when using grid structures like voxels, namely capturing large context information in the structure itself. In general, voxel grid solutions cannot capture large context information, which may be crucial for 3D object recognition and therefore classification. For large objects or particular objects, this may result in low accuracy. State-of-the-art techniques in this field uses Transformer-based and Sparse CNN architectures and one of the relevant paper is Voxel Transformer for 3D Object Detection (VoTr) [5]. VoTr achieves cutting edge results by implementing both local and dilated attention mechanisms to capture both short and long-range context information by combining submanifold and Sparse modules.

These results can be replicated also in the 3D Object Classification field by integrating 3D CNN backbones like VoTr right before a Classification Network; Thinking about future works, also our model may benefit from incorporating techniques that are useful for capturing large amounts of context information as well.

Instead, with this paper, we propose different way to make the neural networks more independent and self continuous of the information on locality through Bagging and 3D cut-out techniques, and we will show an improvement on the overall accuracy on the task.

III. Processing Pipeline

In this part, we explain our methodology, starting with the dataset preparation and giving some insights on our classifiers.

Dataset preparation In the field of machine learning, the availability and quality of training data play a crucial role in determining the performance and generalizability of models. The original dataset we used as starting point, ModelNet10, comes as a clean collection of 3D CAD models for 10 classes of objects. The owner has cleaned and manually aligned the orientations of the models, but they are not scaled nor Independent and Identically Distributed (i.i.d.).

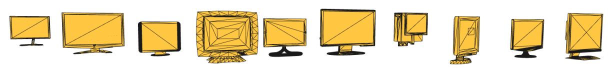
As the initial entries were CAD models, we had to translate them into voxel grids in order to use with 3D Neural Networks.

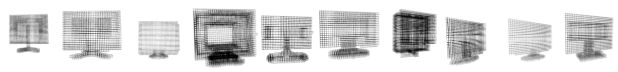
To solve the unbalanced problem in the distribution of the samples, we choose to augment our data with Rotations. Rotations in our task has several advantages, e.g.:

- rotations helps our model to learn a stronger model that can recognize an object from different positions;
- data augmentation helps us to have more data to train our models;

(a.I) Chairs

(b.I) Chairs


(a.II) Monitors


(b.II) Monitors

(a) Example of (2.a.I) Chairs and (2.a.II) Monitors samples from ModelNet10 dataset.

(b) Example of voxelized (2.b.I) Chairs and (2.b.II) Monitors samples of Figure 2.a

Fig. 2.: Examples of Chairs and Monitors samples. Fig. 2.a shows the original CAD models, and Fig. 2.b the voxelized versions of them.

- picking different number of rotations per category, depending on its initial distribution, we are able to balance the samples among the classes. More details in Sec. IV.

We decided to store the rotated dataset efficiently as voxel grids, and start from this for the training. This allowed us to save a considerable amount of computation time, thus leaving only the transformation of those stored voxel grids into Binary Voxels (BV), required by the model.

Inspired by RandomErasing [6] and [7] , the cutout, 3d version of it, is applied to the sample during training. Cutout is a Data Augmentation technique which randomly masking out regions of the input data. In our process, with a probability of 20%, cutout removes a randomly selected 8x8x8 cube from the sample, by putting the voxel contained in this region to 0. This kind of technique encourage the network to learn more robust and general features and reduce overfitting.

Models, Training and Evaluation As our baseline model for the classification, we started with a simple 3D CNN backbone followed by a fully connected layer. Initially, we choose to train the model on a smaller dataset to check if it was powerful enough to overfit, or we needed to increase the depth. Very soon the model overfitted it, thus we stick with the model for further improvements.
In order to make the model learn more general features, we used 3D Cut-Out technique, which improved the results of the baseline model. The best result we get is with our bagging classifier, an ensemble of 5 baseline models trained on different partitions of the dataset. More details on partitioning in Sec. IV and on the classifier in Sec. V.

Fig. 1 shows the training pipeline. We can see that for each weak model, we have a different partition, whereas during the evaluation phase, we used the test partition of the entire dataset.

IV. Dataset

In this section, we start by introducing the dataset used for our experiments, then explain how we processed it to use with our model, and finally how we made it an i.i.d. dataset.

IV.1. Input data

The dataset we used as starting point is ModelNet10, can be downloaded directly from *https://modelnet.cs.princeton.edu/*. It contains 4899 3D CAD models divided into 10 different categories and has an initial splitting of the data into 80% for training and 20% for testing. Fig. 2.a shows some examples of chairs and monitors in the dataset.

IV.2. Pre-processing data

We had to solve mainly 2 problems with the initial dataset:

- the samples were 3D CAD Models, so we needed to convert them into voxel grids for our needs;
- as we already mentioned, the Dataset is not i.i.d., so we decided to augment with rotations to make the categories well-balanced, Fig. 4 in blue shows the initial dataset distribution of the Train folder.

IV.2.1. Voxelization

The CAD models are Meshes, see Fig. 2.a for reference, in Object File Format (OFF) but our model, described in the Sec. V, requires 3D boolean vectors with dimensions of $32 \times 32 \times 32$.

With the help of the Open3d Python library [8], we converted all the meshes into voxel grids after scaling them. As mentioned before, voxel grids of size $32 \times 32 \times 32$ are a good compromise between memory and information loss, and in our case we went from 2.17 GB of CAD models to 311 MB in voxelized version.

Fig. 2.b shows the voxelized samples of Fig. 2.a.

We opted to store the normalized voxel grids directly on disk to reduce computation time during the training phase, as we relied only on the free tiers of GPU provided by Google Colab and Kaggle.

IV.2.2. Data Augmentation

The unbalanced distribution of the dataset may cause models to overfit classes with more samples, thus lowering the performances. Furthermore, in a real-world scenario, we found it very useful to consider that the model should be rotation-invariant, meaning that it can recognize objects regardless of their rotation. Keeping in mind the above and the need to create a robust system, we choose to augment the data with new the rotation along the vertical axis.

The rotations were made by adding multiples of 30 degrees to the initial orientation, thus in total we could have 12 possible rotations. Fig. 3 shows all the possible rotations of the first chair in Fig. 2.a.I. In order to make the classes as equally distributed as possible, we choose uniformly at randomly N different rotations from all the possible ones. N values for each model in Train folder are described in Tab. 1. Meanwhile, we set N = 3 for the Test folder.

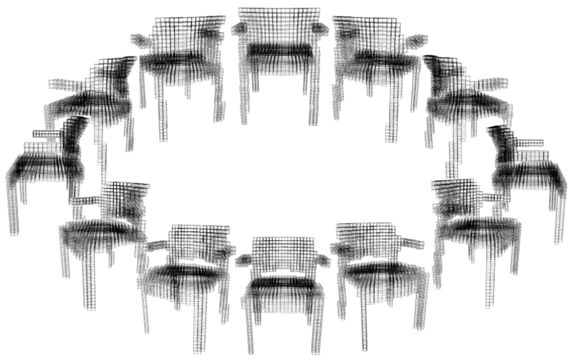


Fig. 3: Example of possible rotation of the first Chairs in Figure 2.b.I

TABLE 1: Categories and their N for Train partition, N is the total number of selected rotations

Rotations	Models
2	bed, chair, monitor, sofa
3	table, toilet
5	desk, dresser, night˙stand
8	bathtub

Fig. 4 shows in orange the distribution after the Data augmentation.

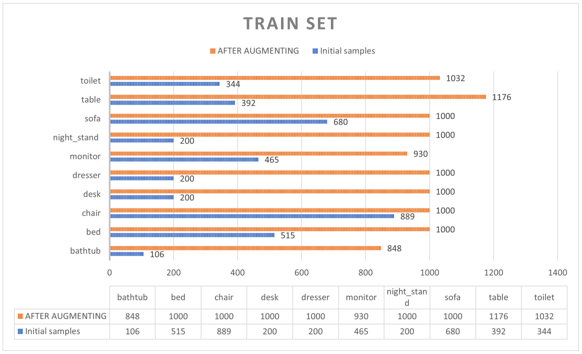
IV.3. Partitioning data

To train the weak models for the bagging classifier, see Sec. V, we created five different subsets with custom rules. The intention was to make each weak model :

- learn well from their part of the dataset (chunk),
- also learn some information from the data of other models.

In details, we:

1) Split tidily the rotated dataset into 25 subsets;



	bathtub	bed	chair	desk	dresser	monitor	night_stand	sofa	table	toilet
AFTER AUGMENTING	848	1000	1000	1000	1000	930	1000	1000	1176	1032
Initial samples	106	515	889	200	200	465	200	680	392	344

Fig. 4: Distributions Before (in blue) and After (in orange) the data augmentation

2) Created 5 initial ordered chunks from the subsets;
3) Copied each subset, of each chunk, into a different dataset w.r.t the initial chunk's dataset.

Tab. 2 shows possible permutations for the datasets.

TABLE 2: Our Bagging datasets compositions

Dataset	Chunk's subsets	Copied Subset
1	01, 02, 03, 04, 05	10, 15, 20, 25 — 24
2	06, 07, 08, 09, 10	01, 11, 16, 21 — 05
3	11, 12, 13, 14, 15	02, 07, 17, 22 — 06
4	16, 17, 18, 19, 20	03, 08, 13, 23 — 12
5	21, 22, 23, 24, 25	04, 09, 14, 19 — 18

IV.4. CutOut

Finally, as explained in Chapter 3, we will perform the cutout on a portion of the dataset, masking out a random 8x8x8 portion of the input data. The probability that the cutout is applied to each sample is 20%.

V. Learning Framework

Bagging and boosting are ensemble techniques very used in machine learning context. Due to the limited resources, we only tried the Bagging, but we can expect similar results with Boosting.

Our final classifier is based on bagging of 5 weak models to predict the class of the object.
Each of the five weak classifier consists of two parts:

- The first part, that we called "3D Backbone", or simply a Convolutional Neural Network (CNN) module;
- followed by a fully connected multi-layer neural network.

V.1. 3D Backbone

In the 3D Backbone, the Input/Output of various layers remain shaped in 3 dimension, preserving spatial relationship. This part can be split into 3 similar blocks, each block is composed by 3D layer as presented in the Fig. 5

The parameters for each of the block of the 3D backbone is reported in the Tab. 3. We also test the Model with different learning rates and the one which seems to bring the best results was: Learning Rate = 1e-3.

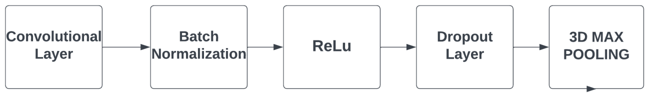


Fig. 5: Single Layer of a 3D Backbone

TABLE 3: Blocks of CNN 3D Backbone

Block	Conv1	Conv2	Conv3
Number of filters	32	64	128
Kernel size	3x3x3	3x3x3	3x3x3
Stride	1	1	1
Padding	1	1	1
Dropout ratio	0.3	0.2	0.2
Max Pool Kernel	2	2	2
Max Pool Strides	2	2	2
Max Pool Paddings	0	0	0

V.1. FC network

The second part of the weak model is composed by 2 fully connected layers. As is common for multi-class classification tasks, the output activation function in the final layer of our models is softmax.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{1}$$

where

- **z** represent the vector given in input to the softmax function
- z_j the j_{th} element of z
- **K** is the number of classes

Since the Convolutional backbone maintains the 3-dimensional shape of the object, in order to make the models make its prediction we flattened the output of backbone in a 1 dimension array, and given in input to the first Fully connected layer, which parameters are reported in the Tab. 4. The softmax function requires that the final output of the models has the number of outputs equal to the number of classes, and so it is.

The loss used for our models is the Cross-entropy loss, which is a common choice for multi-class classification tasks. It measures the dissimilarity between the predicted probability distribution by the softmax function and the true probability distribution of the target variable. The Cross-entropy loss function has the following equations:

$$L(p, y) = -(y * log(p) + (1 - y) * log(1 - p)) \tag{2}$$

Cross-entropy loss is calculated for each example in the training data, and the average loss over all examples is used as the final loss value. We decided to run each of our model for 100 epochs, this choice was made to explore and capture the behavior of the models during the training phase, but how we will see in the next chapter, the models converges with less than 100 epochs.

TABLE 4: Fully connected layers

Layer	Fc1	Fc2
Number of Outputs	128	10
Dropout ratio	0.4	0.0

The final result of the ensembled model is obtained by merging the prediction of all models, we tested two method to take the final prediction:

- **Summing the Probabilities:** We sum the results of the softmax function from all the weak classifiers. The class with the highest sum of probabilities is then chosen as the final prediction.
- **Voting:** In this method, each model in the ensemble makes its prediction, the final result is taken as the most voted class.

We tested both methodologies and in the end we decided to use the first approach, ”sum the probabilities”, for mainly two reasons, the first is that thanks to Softmax function we can explicit a sort of ”belief” of correctness from a model, and we want to use this information for the final prediction, and the second reason is that having only 5 models it implies that the voting system may too easily end up in a tie, which may lead to uncertain results, e.g. 2 votes for class ”bed”, 2 votes for class ”table”, and 1 vote for class ”bathtub”.

In the next chapter we will visualize and comment the results obtained

VI. Results

During the training part we have tried to use both validation accuracy and validation loss as metrics to understand which epoch provides the best model, no noticeable differences were found with the overall accuracy.

Looking at the training epochs, Fig. 6, we can see that the weak models converge quickly: even if we trained our model for 100 epochs, we can see how the convergence is reached in less than 50 epochs.

The results for the weak models fulfils our expectations, and the loss and accuracy on the validation are:

```
    loss: 0.5616 - accuracy: 0.83
    loss: 0.8126 - accuracy: 0.83
    loss: 0.4826 - accuracy: 0.85
    loss: 0.3892 - accuracy: 0.87
    loss: 0.4627 - accuracy: 0.86
```

At the end, the ensembled model shows an overall accuracy of 0.88.

The results show us how the bagging of 5 models increase slightly the overall accuracy of final decision.

Upon exploring the results, see Tab. 5 for reference, we saw that each trained model had difficulties with different object categories, validating our hypothesis that weak models learn different things.

All the trained models has difficulties in classifying similar shaped object like *desk*s, *dresser*s, *night_stand*s, and *table*s. Models usually classify *desk*s as *table*s and *night_stand*s as *dresser*s. This is also valid for the ensembled model, as it

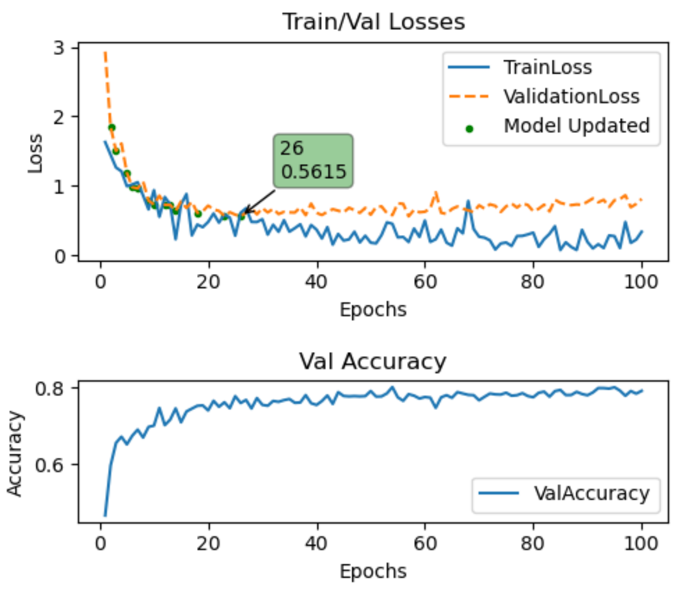


Fig. 6: Training of the Weak Model M0. The upper part shows losses on train and validation sets, and the lower part shows the accuracy on validation

TABLE 5: Weak models and ensembled model precisions on the classes

Class	M0	M1	M2	M3	M4	EN
0:bathtub	**0.98**	**0.98**	0.94	0.97	0.93	**0.98**
1:bed	0.92	0.90	0.92	**0.96**	0.94	0.94
2:chair	0.96	0.91	**0.98**	0.94	0.96	0.97
3:desk	0.62	0.67	**0.80**	0.75	0.65	0.78
4:dresser	0.74	0.73	0.71	**0.83**	0.74	0.77
5:monitor	0.94	**0.97**	**0.97**	0.96	**0.97**	0.96
6:night_stand	0.71	0.75	0.75	**0.81**	0.67	0.75
7:sofa	0.88	0.76	0.81	0.87	**0.90**	0.88
8:table	0.67	0.77	0.73	0.76	**0.81**	0.78
9:toilet	**0.99**	0.90	0.95	0.91	0.98	0.97
ACCURACY	0.83	0.83	0.85	0.87	0.86	**0.88**

has to combine the results of the weaker models, and Fig. 7 confirms it. Fig. 7 shows a matrix representing the predictions of the ensembled model on the test set. The matrix is almost diagonal, which signals that most of the predicted models are correctly classified.

As a benchmark, we tried to train the baseline CNN model with only aligned 3D Objects resulting in an accuracy of 0.90. As expected, the rotations bring a higher level of complexity in classifying the objects. Rotations also make it harder for the models to overfit the dataset.

VII. Concluding Remarks

The solution proposed in this paper provides a good accuracy, moreover we found several advantages compared with a standard CNN model.

First, the memory requirements problem related to the Voxel grid is mitigated. The solution implemented is very scalable,

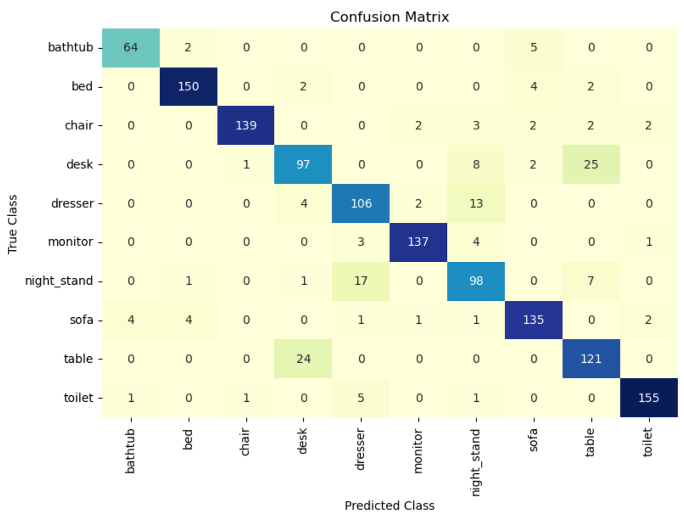


Fig. 7: Confusion Matrix of the Ensembled Model

i.e. there are no specific constraints on the number of weak models, even if it's suggested to choose this number based on the availability of data.

The execution of the whole system is also dynamic and can be adapted based on the potentiality of the computer in which it is executed. The model created can still be improved and enhanced using a larger number of models, and a larger dataset. Beyond enlarging the dataset and the number of models used for the final prediction for the Bagging, we can use another type of ensembling, i.e. Boosting. In our opinion, with Boosting we can further improve the results by paying more attention to wrongly classified classes like desk, dresser, night_stand and table. Furthermore, we would also like to explore the potential of sparse CNN [9] and transformers [5] in classification task, to capture also long-range context information.

During the development of this project we had possibility to develop the models on both, keras and pytorch, and we found that keras is much beginner-friendly than pytorch. Another remark is that python libraries for 3D object manipulation are specialized for meshes and point clouds, but they don't provide full functionalities to work with voxel grid, we tested *open3d* and *trimesh*, the popular ones. Even though Trimesh has more functions to work with voxels, the exportation of the voxel grids is not fully supported, so we used open3d for this project. Finally, we trained our models on Colab and Kaggle, one of the free platforms that provide free GPU for the training, and we found that with colab is easier to connect with drive thus easier to comunicate between the notebook and drive itself for exchanging data. On the other hand, Kaggle provides different types of GPU and TPU. And for our bagging models, we trained the models in parallel, thus saving half of the training time.

References

[1] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," 2019.

[2] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1912–1920, 2015.

[3] N. Sedaghat, M. Zolfaghari, E. Amiri, and T. Brox, "Orientation-boosted voxel nets for 3d object recognition," 2017.

[4] 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.

[5] J. Mao, Y. Xue, M. Niu, H. Bai, J. Feng, X. Liang, H. Xu, and C. Xu, "Voxel transformer for 3d object detection," 2021.

[6] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," 2017.

[7] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017.

[8] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[9] B. Graham and L. van der Maaten, "Submanifold sparse convolutional networks," 2017.

VIII. Individual Contribution

- Simone D'Antimo: CutOut function, Bagging (and voting function), From list of Voxel to Voxel-Grid function, first tries with the CNN models (Keras), Fine-tune of the CNN parameters.

- Harjot Singh: Voxelization of CAD models, Data Augmentation with Rotations, Adaptation from Keras to Pytorch, Fine-tune of the CNN parameters, implementation of ORION neural network, managed the workflow and front end of the final notebooks (Colab and Kaggle).

We both equally contributed to all the sections of the report. We both help each other to perform the relative tasks.