*Research Article*

# 3D object recognition method using CNNs and slicing

Răzvan-Gabriel Dumitru, Sebastian-Antonio Toma
Academic coordinator: prof. univ. Dorian Gorgan

**Abstract:** Three-Dimensional (3D) object recognition has been receiving more and more attention in the field of Computer Vision, as the development of AI algorithms and advancements of hardware components allow for better results nowadays more than ever before. Neural Networks are used to better classify different types of objects, these are being fed with pre-processed 3D objects represented in a 2D domain for better highlighting the important features and for a better performance. In this paper, we propose the strategy of feeding 2D representations of 3D meshes by using different types of slicing to a CNN. In this way, we want to minimize the information loss and the redundancy the data suffers going from a 3D representation to a 2D one. To assess the results, we used Princeton's state-of-the-art ModelNet10 dataset, consisting of 4899 3D objects divided into 10 categories.

## 1. Introduction

As computers nowadays are required to understand as much of our environment as possible to perform different types of tasks in multiple fields, the need for 3D object recognition has risen. 3D object recognition is one of the most import parts in understanding an environment, yet it is a very complicated task for a computer. Various Machine learning algorithms are used in this domain, the best of these being the use of convolutional neural networks (CNNs).

CNNs have proven to be very efficient in 2D image recognition, thus they are also a good choice in 3D object recognition. But generalizing CNNs from a 2D input to a 3D input is the biggest challenge. Applying CNNs directly to a 3D input is very costly in terms of performance, while also not guaranteeing to produce a very good accuracy, due to 3D data being too irregular compared to 2D data.

Three-Dimensional objects are usually represented as 3D meshes or point clouds, both of them being inappropriate for using as inputs for CNNs. Therefore, the problem that arises here is how to properly represent 3D objects to 2D data by minimizing the information loss and the redundancy of the data, while also maintaining the main features of the object that make him be properly categorized. Until now, there are multiple reasonable approaches available of representing 3D objects to 2D data, although none of them guarantees the best results in keeping as much information as possible during the transition.

One of the most popular such representations would be the object's parallel or perspective projection on a 2D plane. However, any type of this known projections loses a lot of information from the original 3D object and concatenating multiple such projection is trivial and produce poor results used together with CNNs. Thus, finding the appropriate 2D

projection of a 3D object to feed to a Neural Network is the focus in this approach of recognition.

There were multiple available approaches and different kinds of projections. But most of such projections required multiple images generated for one object, thus more information is lost, and the learning process is made more difficult for NNs. Previous work also included rotating the 3D objects to obtain its panoramic view and then using the panoramic view projected on a 2D plane, as a single image.

Not all methods based on converting 3D objects to 2D data use Neural Networks to perform recognition on them though. There are some methods using other approaches to identify the objects amongst their 2D projections. One of this type of approaches would be the use of hand-crafted shape descriptors. One of such approach would be using scale invariant feature transforms. It extracts features such as rotation, scaling and translation (RST) invariant, and uses them to match other models based on a similarity measurement. But the rise of deep learning shadows these methods, as NN are more portable and more efficient, after being trained, as the model is used to 'recognize' an object rather than comparing it.

In our paper, we propose a method based on the other type mentioned: using Neural networks, namely Convolutional Neural Networks. As for converting from a 3D object to a 2D representation we propose a new method: slicing. We use slicing through a normalized 3D data set, based on point cloud representation of the 3D object. Before coming to a final approach, we tried other approaches of slicing and even other ways of projections using Principal Component Analysis (PCA). The methods using PCA that we tried seemed to lose a lot of information and not being suitable for this type of dataset that we used.

This paper is structured as follows: the next section (section '2. Related Studies') reviews the already existing work in the area of 3D object recognition; section '3. Methodology' presents the dataset used – ModelNet10 in our research and shows the details of our used methods, the setup and a few details of implementation and the architecture of the NN used; section '4. Results' shows the intermediate results obtained and section 5 represents the conclusion to our project.

## 2. Related Studies

As said before, the previous work is divided in 2 different types of approaches: hand-crafted shape descriptors and Convolutional Neural Networks based methods.

Hand-crafted shape descriptors consist of global and local features. Global features extraction is not appropriate for this type of 3D recognition. 3D local features extraction is more suitable for this type of objects. One of such approach would be using scale invariant feature transforms. It extracts features such as rotation, scaling and translation (RST) invariant, and uses them to match other models based on a similarity measurement. Before using this type of feature extraction, the 3D object is projected on its principal plane using PCA. This is one of the most useful projections as it keeps as much information as possible from the original 3D object. Here a database of scale invariant features of other objects it's used. Basically, this performs matching rather than recognition.

Another studied approach would be decomposing the object into its main component and its salient parts using PCA. This is applied directly on the mesh representation of the 3D object. First, there is applied some smoothing on the mesh, then the protrusion degree of the vertices is computed, obtaining a 4D vector, on which PCA is applied, resulting in a 3D mesh. Neighborhood labeling of the salient parts is obtained with the help of PCA, then differentiation of the salient parts from the main body is done by some Boolean operations. This method is not suitable for this type of dataset, due to the diversity of the salient parts of objects from the same category and major similarity between the main components of objects from totally different categories.

Therefore, Convolutional Neural Networks based methods are more suitable and perform better in 3D object recognition tasks. CNNs have proven to be very efficient in many fields, especially in Computer Vision tasks. Having a good performance in 2D image recognition, is what recommend them as good choices in 3D object recognition.

Many researchers started to feed to CNNs volumetric representations of 3D objects, using voxelization base methods. Voxelized shapes can be seen as an extension of the 2D pixels, making data seem more regular for CNNs. But this type of volumetric representation seems to limit the resolution of the final product, thus losing information and accuracy. However, the number of voxels can be increased so that this representation can match the resolution of a 2D image representation. By doing this, the number of voxels needs to be large, and the representation tends to be sparser. This results in lower performance of the network and lower efficiency of the use of memory.

Another popular approach proposed by Shi et al. is DeepPano, which use the panoramic view obtained by projecting the 3D object on a cylindrical surface. Then these 2D representations are fed to multiple types of CNNs, where they have introduced a novel layer, namely row-wise max-pooling (RWMP) to enhance the typical pooling layer in CNNs for obtaining robustness for rotation.

More studies have evolved since DeepPano, producing astonishing results. One of the leading studies in this approach is the one proposed by Zheng et al., obtaining 89.8% accuracy

on the ModelNet10 dataset. In this study, they use the panoramic view of the object proposed by DeepPano, together with an improved Oversampling method. In addition, they propose a novel method for overcoming the shortcomings found in the DeepPano method, namely rotation expansion. In this method they basically unfold the cylinder at a series of different reference angles, generating several copies for each 3D shape. In this way, the model will get rotational invariance as features of the 3D object that are robust to rotation are also included.

In this paper, we chose a method based on Convolutional Neural Networks, combined with slicing.

## 3. Methodology

In this section we present the dataset used, the setup used for the project, the various methods we tried before coming to a final one and the final method and its implementation.

*3.1 Dataset*

The dataset we used is Princeton ModelNet10, which is one of the most used datasets by researchers in computer vision, computer graphics, robotics, and cognitive sciences. It is made from a comprehensive clean collection of 3D CAD models for objects stored as .off files. Every .off file contains the vertices and the faces (as triangles), which correspond to a description of the mesh model. The dataset available online contains CAD models from 10 categories used to train the deep neural networks. Training and testing split is already included in the file. The training set includes 3991 models, and the test set includes 908 models. The CAD models are completely cleaned inhouse, and the orientations of the models (not scale) are manually aligned by the providers. The complexity of the objects is spread out, from a simple rectangular object to a very detailed furniture. Thus, the size of a file representing an object ranges from 4KB to ~23000KB. The 10 categories, the number of objects per each category, an example viewed on an online 3D viewer and the train-test split ratio are provided below:

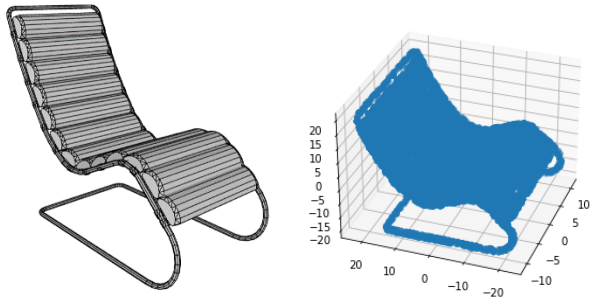
- Bathtub: 156 items – 32% for test
- Bed: 615 items – 16.2% for test
- Chair: 989 items – 10.1% tests
- Desk: 286 items – 43% for testing
- Dresser: 286 items – 43% for test
- Monitor: 565 items – 17.7% for test
- Night stand: 286 items – 43% for test
- Sofa: 780 items – 12.8% for test
- Table: 492 items – 20.3% for test
- Toilet: 444 items – 22.5% for test


*3.2 Methods*

*3.2.1 Point cloud transformation*

Objects are given in .off files. Every .off files is a representation of the 3D object as a 3D mesh. The file contains the number of vertices 'v' and the number of triangles 't' at the beginning. Then 'v' lines of vertices follow, a vertex being represented in (x, y, z) coordinates. The next 't' lines contain 4 integers each. The first integer is the number of vertices the polygon has, which is always 3 in our case as we work with 3D triangular meshes, so every polygon is triangle. The next 3 integers represent the index of the vertices that forms the triangle face. Therefore, a mesh is a representation made from a series of triangles, and the position of each triangle vertex is given as (x, y, z) coordinates. But, for our approach, we found that point cloud representation of a 3D object is more favorable. Triangular meshes are convenient to be used in many computer graphics task, but for 3D data processing and extracting information from a 3D form to feed to a NN in the end, point cloud representation is more suitable.

First, the meshes are converted to 3D point clouds for further manipulation with a sample rate of 8192, that means each object is now represented as 8192 points in (x, y, z) coordinates. The conversion is done by randomly sampling points on the surface of the mesh.



*3.2.1 Normalization*

After the 3D object is represented as a 3D point cloud, but before representing the data in a 2D space, a normalization process is applied to the 3D data. A process of normalization is applied to every model as follows: the 'x' and 'y' axis are normalized according to the greatest range among all the three axes of the given object, while the 'z' axis is normalized according to its own range, i.e. it is the fixed axis, and then the points are scaled:

$$P.x := \frac{P.x - \min_{P \epsilon M}\{P.x\}}{\max\{range(Ox), range(Oy), range(Oz)\}},$$

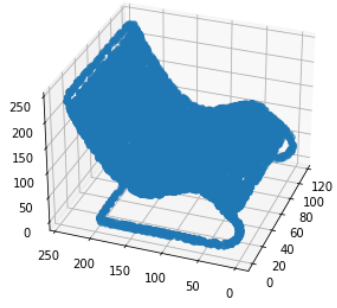
$$P.y := \frac{P.y - \min_{P \epsilon M}\{P.y\}}{\max\{range(Ox), range(Oy), range(Oz)\}},$$

$$P.z := \frac{P.z - \min_{P \epsilon M}\{P.z\}}{range(Oz)},$$

$$P := P * 240, \forall P \in M.$$

Where P represents a point in the object model M.

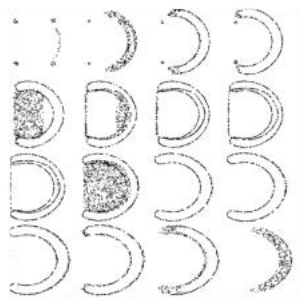
The models need to be normalized and scaled, because of the vast diversity of sample objects. As said before, their size varies a lot, and after converting them from 3D triangular meshes to 3D point clouds, their range it's still very different amongst them. This would make the projection process though and would produce very different images, even for objects from the same category, which would negatively impact the learning process for the Neural Networks. The 'z'-axis is 'normalized' separately from the other two axes, for further assurance that the 2D Simple Slicing (slicing done through the 'z'-axis) is properly done. Also, all the objects are represented 'standing' as in real life, so their height corresponds to the 'z'-axis.




*3.2.2 2D Simple Slicing*

After the data is normalized, the slicing method is applied on it. The slicing is done on the 'z'-axis, by dividing the object in exactly 16 different parts. This is ensured by the normalization from the previous step, where the 'z'-axis was independently normalized from the 2 other axes, it was the fixed axis. Each slice basically represents the view from the top of the object from different heights. This type of slicing can be viewed as slicing through the height of the object. Every point in the original 3D point cloud representation is 'assigned' the closest slice of one of the 16 slices. Also, instead of 'assigning' one point, the point is 'split' in 3 points to increase the clarity of the final slice by a factor of 3. Therefore, every slice is a set of points representing a picture in black and white format, where every 'point' has a value of '0' or '1' for further optimization of the memory, without reducing sharpness of the image.

In the end, these 16 slices / views are concatenated together in a 4x4 matrix and used further as a one black and white image. The slices have some additional space, so each respective slice can be clearly distinguished from the others, even by the human eye, to make the Neural Networks properly learn the difference between each slice. An example of the final output and the illustration of the process of slicing can be viewed below.



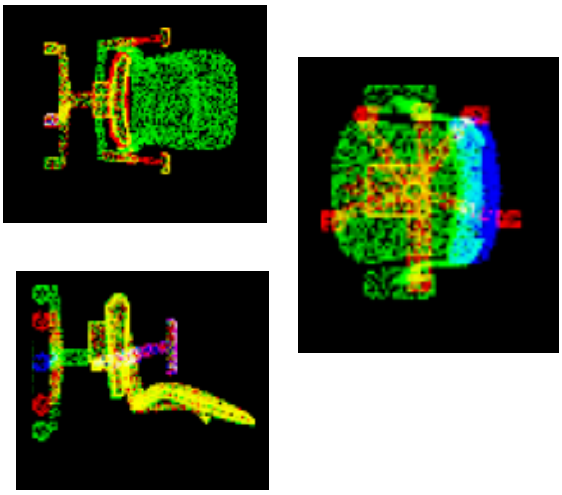
*3.2.3 2D Multi-axis Slicing*

In this section, another method of slicing is proposed, for a better information retention from the 3D normalized point cloud representation to a 2D projection. This is the final method used to feed to the Convolutional Neural Networks.

Here, the slicing is done now on all three axes instead of only on the 'z'-axis as in the previous method. The process of slicing is somewhat similar to the one described before but generalized to every axis and applied to all three axes ('x', 'y', 'z'). The object is divided in maximum 12 different parts / slices. Each slice basically represents the view from the top of the object from different heights for the 'z'-axis, like in the previous method and for the 'x' and 'y'-axis analogously the

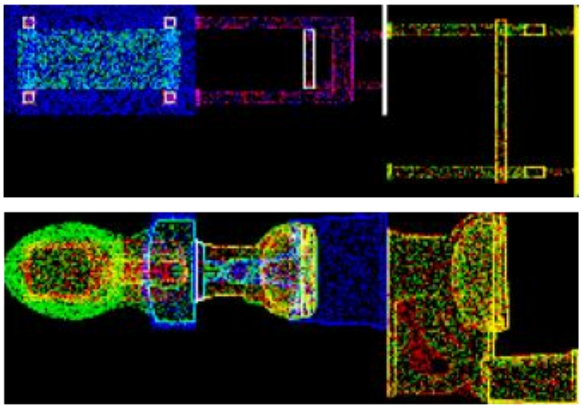
same. Each slice is the view of how the object is seen according to that axis and to corresponding height / width / length of the object. Every point in the original 3D point cloud representation is 'assigned' the closest slice of one of the 12 slices. The same approach of increasing the clarity of the resulting picture as in the previous method (i.e., before 'assigning' the point to a slice, the point is 'split' in 3 points to increase the clarity of the final slice by a factor of 3) is discarded in this method. The reason will follow up after the presentation of the next steps taken to obtain the final 2D image with this method.

After applying the slicing method on each of the three axes, a set of slices / views is obtained for every axis. Instead of concatenating these slices like in the previous method, we overlap / merge these slices on every axis. Here, the resulting picture will no longer be in black and white, a 'point' in the picture will no longer have the values '0' or '1'. The resulting picture will be a colored picture, where each point will have a color, representing to some extent the depth or the density of the points in the given area. This will consist in a 2D projection of the 3D object on a given axis. The resolution of the picture will be good enough, so there is no need of 'splitting' each point in three, in fact this would only overload and make the feature distinction harder for the CNN.



Therefore, we obtain a colored picture for every axis, resulting in three pictures, each one representing the merge of all the slices of a given axis. To make a single picture that is going to be fed to a CNN, we concatenate all three pictures, by arranging them in the same 'row'. Below, we can see an example of how the final input picture to the CNN would look like for a given 3D object.

Initially, adding some padding between every projection of the axis was considered. The padding consisted of some empty 'white' space, so the CNN model could better distinguish amongst the three axes. In the end this lead to carrying over more useless information, consuming redundant space, without improving the ability of the CNN to distinguish amongst axes, so it ended up being removed from the final version, as there is no need to it.



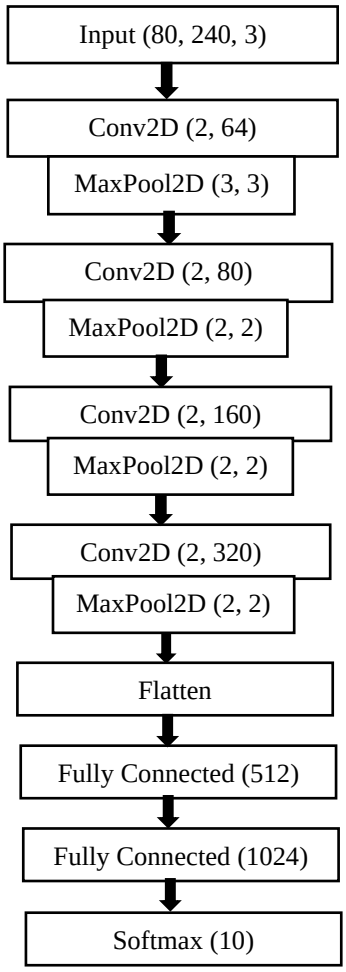
## 4. Experimental Results

*4.1 Implementation*

In our implementation, we used Python for both preprocessing the data and for the CNN. The dataset was downloaded from Princeton's ModelNet official website, and it was already split in train and test sets by them, the ration being described above in the 'Dataset' section. Every category had its own train-test split. The data is given as .off files, each file corresponding as one 3D object, represented as a triangular mesh. For reading and manipulating the triangular mesh the 'trimesh' library is used. Each object read in memory as a triangular mesh is then converted to a 3D point cloud, by sampling through the mesh, resulting in a set of 8192 (x, y, z) points for each object. In this way, all objects have the same size, the data being more consistent. Therefore, 2 lists of sizes 3991 and 908 are obtained, corresponding to the train and test dataset, each one containing an object of the shape (8192, 3). After applying the normalization process and slicing process described above, each object will be the shape of (80, 240, 3), which represent three 80x80 RGB pictures concatenated in a row.

These pictures corresponding to the train set are fed to the CNN, and the ones corresponding to the test set are used to test the model after it has been fitted. The CNN was built using TensorFlow and Keras frameworks. We have 4 convolutional layers with 64, 80, 160, 320 feature maps respectively, and kernel size 2 for each of them. After each convolutional layer, a corresponding max pooling layer is inserted, first a 3x3 one, then 2x2 layers. This type of

convolutional architecture has been proven to be very efficient in learning from such 2D RGB images. After the convolutional layer, a flatten layer is inserted for connecting to the next layers, two fully connected layers with ReLU activation with 512 and 1024 respectively hidden units. Then, the final layer is a fully connected Softmax layer with 10 classes. Each class outputs a probability of the given object being in this class's 'represented' category, then the class with the highest probability is chosen as the final prediction. We used the accuracy of the predictions as the metrics and 'Adam' as our model's optimizer. For the loss function we used Sparse Categorical Crossentropy. The architecture of our network is similar to the one used by Zheng et al. and it can be viewed below.




*4.2 Setup*

To evaluate the results, our model was trained on the Princeton's ModelNet10 dataset consisting of 10 categories of objects. The dataset can be downloaded from Princeton's ModelNet official website, where there is also a ModelNet40 dataset, a dataset similar with ModelNet10 but with 40 categories instead of just 10.

We implemented the CNN and run the experiments on a virtual machine provided by Technical University of Cluj-Napoca, with the following specifications: Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz, 16 cores, 128 GB RAM, 256GB SSD, running on Ubuntu 20.04 Desktop. The code was run from a local IDE (JetBrains DataSpell), from which we connected to the virtual machine using a Jupyter Notebook server.

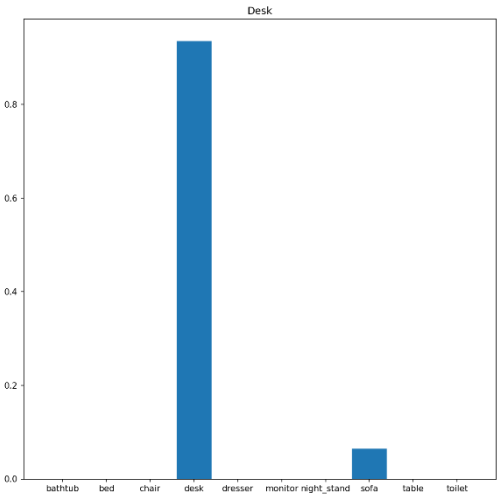
| Layer | Output Size | Parameters |
|---|---|---|
| Conv2D | (79, 239, 64) | 832 |
| MaxPooling2D | (26, 79, 64) | 0 |
| Conv2D | (25, 78, 80) | 20560 |
| MaxPooling2D | (8, 26, 80) | 0 |
| Conv2D | (7, 25, 160) | 51360 |
| MaxPooling2D | (3, 12, 160) | 0 |
| Conv2D | (2, 11, 320) | 205120 |
| MaxPoolin | (1, 5, 320) | 0 |
| Flatten | (1600) | 0 |
| Dense | (512) | 819712 |
| Dense | (1024) | 525312 |
| Dense | (10) | 10250 |

*4.3 Object classification*

All the ten categories and the number of elements in each category, as well as the number of objects in the train set and the test for each category are provided in the 'Dataset' section above. The train-test split ratio is also provided above, it ranges from 10.1% to 43% of the objects corresponding to the test set. Also, there is a significant discrepancy between the number of objects in each category, from 156 objects in the 'bathtub' category to 989 objects in 'chair' category, this dataset can be considered as an unbalanced one. This means that the model will benefit the categories with more representants and will disadvantage the one with fewer representants. But this is not exactly the case, as the dataset is well-aligned and the orientation of the 3D model is highly consistent, as the dataset was manually aligned by Princeton's providers. Therefore, the complexities of the test and training completely match. So, the features learnt

from this training set are enough to recognize the objects in the test set with a good performance.

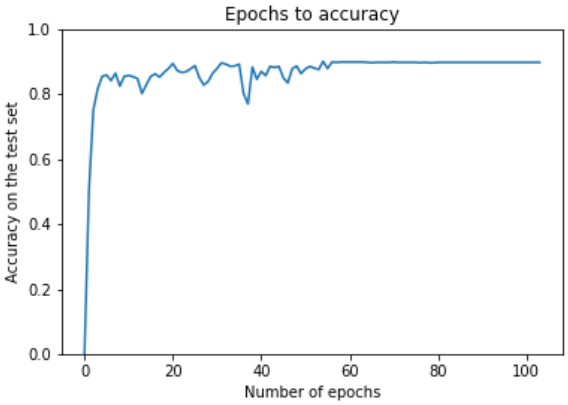
The output of the model is a list of probabilities of an object being in each category, these are called the predictions. An example of such a prediction on the test set is given below, where see the model categorized the respective given object as approximative 90% to be a desk and less than 10% to be a sofa (the model was right in this case).



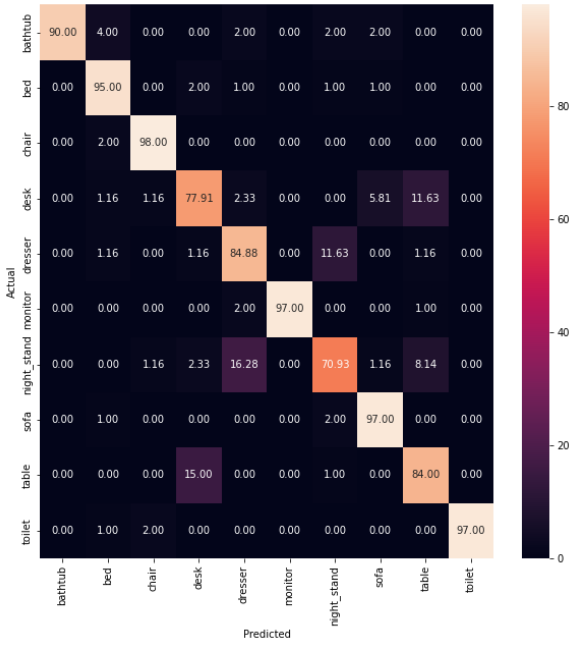

The final performance is represented by the average on each category accuracy. The final accuracy on the ModelNet10 dataset for our model computed on the test set is **89.98%**, which is higher than Zheng-shape, who got 89.80% on ModelNet10. We compare our method with Zheng-shape, DeepPano (both of them using the Panoramic View approach combined with CNN) and 3D-ShapeNet (which uses volumetric shapes representation and CNNs). Also, we include our first approach in the comparison that uses the 2D Simple Slicing method that was fed to a slightly different CNN from the final one. The comparison can be seen in table below.

| Method | ModelNet10 |
|---|---|
| Simple Slicing | 86.2% |
| Multi-axis Slicing | **89.98%** |
| DeepPano | 88.66% |
| 3D-ShapeNet | 83.54% |
| Zheng-shape | 89.80% |

The model learns the features of the dataset fairly well even from the first epoch, as it got around 81% after the first epoch, during training. The model oscillates between 77% and 89% during the first 60 epochs. From the 60 epochs onwards it stabilizes around 89.98%, which is the final accuracy on the test set. We fitted the model for around 100 epochs, which took around 30min to run on Intel(R) Xeon(R) Gold 6230 CPU. This is one of the fastest models from the ones compared in this study. Below there is the evolution of the accuracy during the training epochs.




To see the accuracy on every distinct category on the test set, the Confusion Matrix was computed, which uses the predictions of the model on the test set.



## 5. Conclusion

In this paper we propose a novel method of 3D object recognition, based on Princeton's ModelNet10 dataset. Our approach is different from other approaches by means of preprocessing the data before being fed to a CNN. We proposed a method based on slicing (Multi-axis Slicing) on all three axes, that is both efficient and accurate. The 2D representation obtained this way from the 3D object is given as three concatenated pictures, each picture representing the slices through an axis. We used CNN for the model, because they are suited for 2D image recognition task, which is what the problem of 3D recognition arrives to, after converting the 3D data to a 2D projection. In this way, we obtained a good accuracy on the test set, which is 89.98%. We surpassed DeepPano by 1.32%, Zheng-shape by 0.18% and 3D-ShapeNet by 6.44%. Our method works on 3D triangular meshes by converting them to point clouds, thus it can be extended easily to work directly on 3D point clouds, which can be in the form obtained from the LIDAR sensor. Our study can be used in many Computer Vision tasks, in self-driving cars, AR/VR or computer graphics or for further research and future development.

### Acknowledgements:

This research was supported by the Technical University of Cluj-Napoca by providing us with a virtual machine to run our model on.

### Data availability:

https://modelnet.cs.princeton.edu/

### References (only links for now) –TO DO--

[1] An Improved 3D Shape Recognition Method Based on Panoramic View (hindawi.com)
[2] https://www.researchgate.net/publication/220067064_Protrusion-oriented_3D_mesh_segmentation
[3] https://www.researchgate.net/publication/220588094_Principal_Component_Analysis-based_Mesh_Decomposition?enrichId=rgreq-b425b15be8ed506b62b423c62a90dc34-XXX&enrichSource=Y292ZXJQYWdlOzIyMDU4ODA5NDtBUzoxMDM0Mzk3NjU4Njg1NTZAMTQwMTY3MzM3MDcwNQ%3D%3D&el=1_x_2&_esc=publicationCoverPdf
[4] https://www.researchgate.net/publication/222414810_3D_CAD_model_matching_from_2D_local_invariant_features?enrichId=rgreq-7e03bb09bed531ac09bb81925778b1c1-XXX&enrichSource=Y292ZXJQYWdlOzIyMjQxNDgxMDtBUzo1NTQ0MjUzNzQ5ODYyNDBAMTUwOTE5NjcxODI0Mw%3D%3D&el=1_x_2&_esc=publicationCoverPdf
[5] https://github.com/timzhang642/3D-Machine-Learning
[6] https://www.kaggle.com/balraj98/pointnet-for-3d-object-classification-pytorch
[7] https://keras.io/examples/vision/pointnet/
[8] https://openaccess.thecvf.com/content/ICCV2021/papers/Xiang_Walk_in_the_Cloud_Learning_Curves_for_Point_Clouds_Shape_ICCV_2021_paper.pdf
[9] https://paperswithcode.com/paper/polynet-polynomial-neural-network-for-3d
[10] https://paperswithcode.com/paper/orientation-boosted-voxel-nets-for-3d-object
[11] https://arxiv.org/pdf/1707.01083.pdf