NurbsNet: A Nurbs approach for 3d object recognition

Escalona, Felix
felix.escalona@ua.es

Viejo, Diego
dviejo@ua.es

Fisher, Robert B.
rbf@inf.ed.ac.uk

Cazorla, Miguel
miguel.cazorla@ua.es

March 27, 2020

Abstract

With the emergence of low cost 3D sensors, the focus is moving towards the recognition and scene understanding of tridimensional data. This kind of representation is really challenging in terms of computation, and it needs the development of new strategies and algorithms to be handled and interpreted.

In this work, we propose *NurbsNet*, a novel approach for 3D object classification based on local similarities with free form surfaces modeled as Nurbs.

The proposal has been tested in ModelNet10 and ModelNet40 with results that are promising with less training iterations than state-of-the-art methods and very low memory consumption.

1 Introduction

Scene understanding is one of the main challenges for autonomous robots. In this respect, object recognition is one of the key tasks to be performed to accomplish this quest [1].

Until 2012, all the proposals submitted to the *Large Scale Visual Recognition Challenge*, an object recognition challenge with ImageNet images database [2], were based on handcrafted features and classical classifiers. However, that year appeared the first *Deep Learning* approach [3], that outperformed the previous techniques.

Since that time, the research focus changed to *Deep Learning*, with a great and rapid evolution and an enormous set of new architectures and learning strategies. For the 2D object recognition task, several approaches have achieved high classification rates, as explained in the survey [4]. However, 3D object recognition is still an underexplored research field compared to its counterpart in 2D.

In this work, we propose *NurbsNet*, a novel approach for 3D object recognition based on the search of local similarities between the object and internal free form surfaces.

This paper is structured as follows. Section 2 reviews the state of the art for 3D object recognition, focusing on the new deep learning methods. Then, Section 3 describes the fundamentals of our proposal, with a brief introduction to Nurbs surfaces, an explanation of our proposed similarity metric and the description of our end-to-end architecture. Next, Section 4 reports experiments carried out with ModelNet10 and ModelNet40 and the methodology to test the proposal. Finally, Section 5 draws some conclusions about the project and establishes future lines of research.

2 Related works

In this section, we present a review of recent deep learning methods for 3D object classification. Following the classification presented in [1], we divide the methods in terms of the most used data representation.

- **Pointcloud representation**. These techniques use the 3D data represented as a unordered set of points in 3D space. They extract spatial features directly using nearest-neighbours and radius search, so it can be computationally expensive. *PointNet* [5] calculates features over the points and apply transformations to the data until it builds a global feature vector, that feeds a neural network for classification or segmentation purposes. *PointNet++* [6] is an extension of the previous method that learns hierarchical features from the points, introducing layers of sampling and grouping. Our work can be included in this classification, as it works with the point clouds directly.

- **Voxel representation**. In this case, the input data is represented as a discretization of the space around the data as an approximation of the original form. Every voxel usually contains a 0 or 1 indicating the presence of points, or a value stating the density of points that lie inside it. The original proposal from the creators of ModelNet, *3D ShapeNets* [7], represents the data as a cubic voxel and apply 3D convolutions with restrictions to obtain the vector representation. Further works take this idea with variants. *VoxNet* [8] applies a 3D *Convolutional Neural Network* to this volumetric representation for classification purposes. However, not only *CNNs* have been used, but also novel deep learning architectures: *Vconv-dae* [9] employs a convolutional denoising auto-encoder as a feature learning network, [10] uses a *Variational auto-encoder* and [11] a *Generative Adversarial Network*. The main issue of these techniques is the lack of precision when discretizing an object, which can lose fine details, and the enormous space requirements of the 3D convolutions in memory.

- **Image slices representation**. This family of techniques does not use the 3D data directly, but they transform it into 2D projections, *slices*, that can

be processed using *Convolutional Neural Networks* for 2D images. These are the most common approaches, that use a single or multiple views of the object to feed a *Convolutional Neural Network*, as presented in [12]. Some works have focused on grouping views and training a boosting classifier to improve their performances [13]. Other approaches are based on the selection of the best views of the object to make the inference, as presented in [14], that uses 3 orthogonal views that feed 3 independent neural networks. This group of techniques has the best classification performances, based on the ModelNet benchmark [15], but they need a full reconstruction of the object in order to generate multiple views, so they are not so reliable in real-world applications when dealing with occlusions and partial views.

3 Approach

Our method receives a point cloud as input and generates a probability distribution of labels for the received object. Our proposal introduces two new approaches: a *Nurbs layer* that learns free form surfaces and calculates the similarity between every Nurbs and local shapes of the input cloud, and a *Voxelization layer* that uses a spherical grid that selects the best activations and generates a feature vector to feed the *Fully Connected layer*.

In Section 3.1 we explain the fundamentals of the Nurbs surfaces, the fitting process and the similarity calculation. In Section 3.2, we present the architecture of our proposal with an extensive explanation of our novel layers.

3.1 Nurbs surfaces

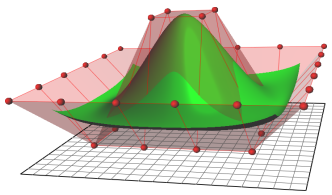

Figure 1: Representation of a Nurbs surface. Extracted from [16]

We have chosen Nurbs to represent freeform surfaces in a parametric form. This mathematical model has been widely used by the CAD industry and the academic community as a 3D representation due to its expressiveness and relative simplicity. This kind of representation allows modifying the local geometry of an object by moving a few control points, so it favors surface optimization.

A Nurbs surface is represented by Equation 1, where $P_{i,j}$ are the 3D control points, $w_{i,j}$ the weights for every control point and $N_{p,q}$ are the normalized B-spline basis functions of degree p. Every $N_{p,q}$ function affects the surface in a limited range, defined by the knot vectors [17].

$$S(u,v) = \sum_{i=1}^{k}\sum_{j=1}^{l} R_{i,j}(u,v)\mathbf{P}_{i,j}$$
$$R_{i,j}(u,v) = \frac{N_{i,n}(u)N_{j,m}(v)w_{i,j}}{\sum_{p=1}^{k}\sum_{q=1}^{l} N_{p,n}(u)N_{q,m}(v)w_{p,q}} \quad (1)$$

3.1.1 Nurbs fitting

Nurbs fitting consists of the reconstruction of a complete surface from a limited set of 3D data, calculating its parameters iteratively.

The simplest method to minimize the distance between the pointcloud and the generated surface is *least mean squares minimization*. The measure for the distance can be the Euclidean distance, known as *point-distance minimization* (PDM), the *tangent distance minimization* (TDM) [18] or the *squared distance minimization* (SDM) [19].

The surface is initialized using *Principal Component Analysis* (PCA), using the plane formed by the two eigenvectors with the greatest eigenvalues of the data, where the initial control points will lie. Then, in every iteration the distance between the points and the surface is calculated according to some distance criteria (as mentioned above) and the control points are updated in the direction where the distance decreases.

The fitting method was proposed in [20] and the implementation has been taken from the *Point Cloud Library*, carried out by Thomas Moerwald [21].

3.1.2 Similarity metric between two Nurbs surfaces

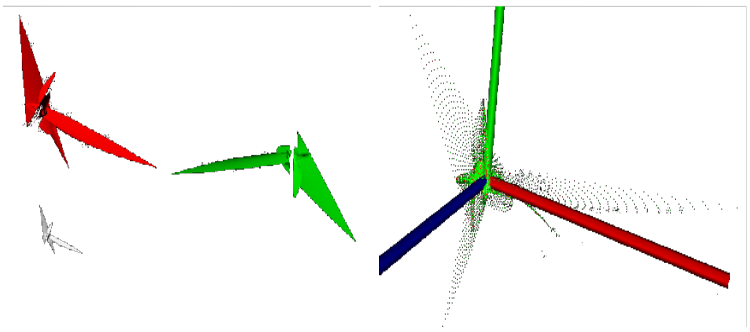

Figure 2: Red, green and grey nurbs surfaces in the left have the same shape but different scale, rotation and translation. They are represented in the right with the same sample points in euclidean space after the use of the techniques explained in Section 3.1.2

There are some works like [22] and [23] that propose a similarity measure of Nurbs using the Nurbs Warping method [17] for image retrieval purposes.

However, these techniques are only applicable to 2D images because the shape of a 3D object is far more complex and cannot be approximated by simply *fitting their borders.*

To solve the similarity problem between Nurbs surfaces in 3D, we propose a pointwise approach, using the following method:

1. **Sampling of the surface**. Using the Nurbs equation and iterating over the parameters u and v we can obtain a set of points from the surface. There are some studies [24] and [25] that investigate this topic in order to choose the proper parameters that generate the points that better represent the surface.

2. **Normalization**. We want identical surfaces at different scales to be able to have minimum distance, so we have to make the sample point representation invariant to scale. First, we normalize for translation by aligning the centroids of the two surfaces. Then we calculate the median distance of the points from the centroid. Then, we carry out the normalization step by dividing every coordinate by the median. Dividing by the maximum distance could be an alternative, but the median is less affected by noise.

3. **Principal Components Analysis**. Similar surfaces with equivalent but rotated principal axes should have lower distances. In order to achieve the rotation invariance, we do *Principal Component Analysis* (PCA) on the pointcloud to obtain the axis that capture the maximum variability, and apply the transformation matrix that changes its basis, as explained in [26]. *PCA* does not ensure the orientation of the axis (only the direction), so we save the four possible permutations of the sign of two principal vectors, because the third axis is calculated via vector product. We will use the four representations of every surface to calculate the distance to another figure.

4. **Distance measuring**. We calculate the distance of every sampled point of the first surface to the other surface. This distance can be measured in several ways, with the euclidean distance to the nearest neighbour in the other surface the most common, simple, and good enough for our purposes, as shown in Equation 5. The resulting distance between the surfaces is the maximum of the two minimum unidirectional distances (Equation 3). Finally, the similarity is calculated from the resulting distance using Equation 2. Notice that in Equation 4, when we calculate the distance from one surface to another, we only iterate over the permutation of the axis of one of the surfaces to exploit the symmetries of the representations and reduce computational costs.

As the output of this process, we obtain a similarity score that will be 1 when the surfaces represent the same shape, and will decrease as their differences grow. In this way, we have represented the points of the Nurbs with scale, translation and rotational invariance.

$$similarity(s1, s2) = 1 - d(s1, s2) \tag{2}$$

$$d(s1, s2) = max(d\_unidir(s1, s2), d\_unidir(s2, s1)) \tag{3}$$

$$d\_unidir(s1, s2) = \min_{\forall i \in s1} d\_sing(s1_i, s2_1) \tag{4}$$

$$d\_sing(s1_i, s2_j) = \frac{1}{|s1_i|} \sum_{\forall p_i \in s1_i} d(p_i, n\_neigh(p_i, s2_j)) \tag{5}$$

3.2 Network architecture

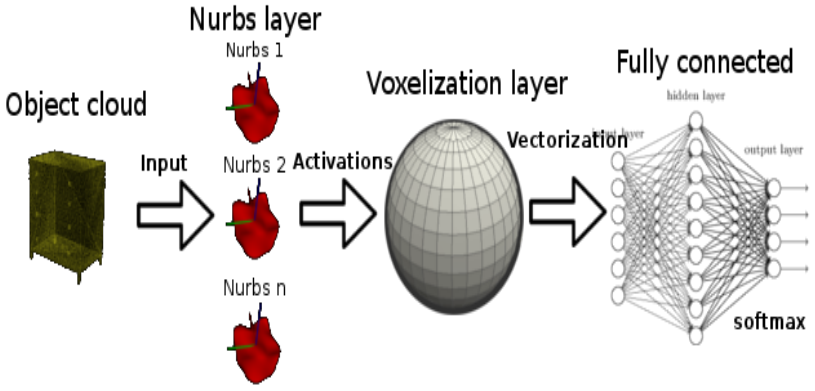

Figure 3: End-to-end network scheme of out proposal

The scheme of our end-to-end architecture is depicted in Figure 3. The *Nurbs layer* receives the point cloud of the segmented object. It calculates the activations for every point with every internal Nurbs in the layer and sends this information to the *Voxelization layer*. This layer discretize the space around the object with a spherical mesh divided in prism sections, and calculates the best activation for every Nurbs that lies in every sector. Finally, it flattens the output as a vector and sends it to the *Fully connected layer*, which calculates a classification probability for every class.

The *Nurbs layer* is explained in Section 3.2.1 and the *Voxelization layer* in Section 3.2.2.

3.2.1 Nurbs layer

This layer is directly connected with the pointcloud input of the network. As a preprocessing step, we perform a normalization of the cloud similar to the one explained in Section 3.1.2 for Nurbs. Additionally, we can apply a voxel grid

filter to reduce the dimensionality and favour the generalization of the learning process.

The Nurbs layer contains a set of internal Nurbs and replaces the classical convolutional layer of a *CNNs* by calculating similarity scores between surfaces. It is important to notice that this layer can work with unordered point clouds of variable size.

The main task carried out by this layer is the calculation of the similarity score between the internal Nurbs and every local Nurbs of the input, following these steps:

1. **Fit local Nurbs for the input**. For every point in the input point cloud, extract its neighbors using a fixed radial neighbourhood. Then, apply the fitting process explained in Section 3.1.1. The result of this process is a local Nurbs for every point in the cloud.

2. **Calculate activations**. For every local Nurbs generated in the previous step, calculate their similarity score with all the internal Nurbs of this layer, as explained in Section 3.1.2.

This layer has some hyperparameters that must be set: the radius of search for neighbors, the number of internal Nurbs and their number of control points. Furthermore, there are some strategies for initializing the Nurbs control points: random initialization (using a normal distribution), initialization from geometric surfaces and initialization from real surfaces.

3.2.2 Voxelization layer

This layer receives the output from the *Nurbs layer*, explained in Section 3.2.1, as a vector of activations of every point in the point cloud with every internal Nurbs in the *Nurbs layer*. The main goal of this layer is to generate a descriptor that can be suitable for the input to the fully connected neural network. In this case, we have chosen a discretization technique of the space around the point cloud, but in a different manner than usual.

Many approaches like [27] and [28] build a volumetric ocupation grid, in form of cube voxels, to represent the shape of the figure. However, point clouds are usually hollow in their center, so this technique is not efficient for our purposes as it produces so many holes in the final vector representation.

In this case, we propose another technique of discretization, based on a spherical coordinate space, following these steps:

1. **Convert points into spherical coordinates**. Calculate the centroid of the pointcloud and apply a translation to convert the centroid to the origin of coordinates (0,0,0). Convert the points from cartesian to spherical coordinates following Equation 6.

$$\begin{gathered} R = \sqrt{x^2 + y^2 + z^2} \quad \phi = \tan^{-1}\left(\frac{y}{x}\right) \\ \theta = \cos^{-1}\left(\frac{z}{\sqrt{x^2+y^2+z^2}}\right) \end{gathered} \tag{6}$$

2. **Normalize**. Normalize the radius component R dividing it by the largest radius of the point cloud.

3. **Discretize**. Discretize every spherical coordinate using the defined hyperparameters of resolution for every coordinate. Radius R is defined in the range $[0, 1]$, θ in the range $[0, \pi)$ and ϕ in the range $[0, 2\pi)$. It's recommended to use a resolution of 1 for the radius component to minimize the holes of the representation. We divide the space into prism sections according to the provided resolution, similar to the techniques used in 3D descriptors like 3D Shape Contexts [29] or Unique Shape Context [30].

4. **Select activations**. First, we identify which points lie in every prism section. Then, for every data point in that section and every internal Nurbs, we look for the best activation based on some criteria like maximum, minimum or average activation. This selection criteria does a similar task to the pooling layers of the *classical* convolutional networks.

5. **Build representation** Using the best activation of every Nurbs in every prism section, build a linear vector with these values given an order of sections.

As an output of this layer, we have a linear representation of the input data that represents the local similarities of the point cloud with the internal Nurbs of the *Nurbs layer*. This vector of characteristics is fed to the fully connected layer to perform the classification of the object.

4 Experiments

In order to test and compare our architecture with other state-of-the-art methods, we have chosen the *Princeton ModelNet* dataset [7]. It provides an extensive collection of 3D CAD models of objects. There are two versions of the dataset, *ModelNet10* and *ModelNet40* (not aligned and aligned version), that have been widely used as benchmarks for 3D object classification. *ModelNet10* offers a set of more than 5000 CAD models from 10 different categories manually aligned, and divided into training and test sets. Following the steps explained in [27], we converted these models into *Point Cloud Data* (PCD) clouds, compatible with the *Point Cloud Library* (PCL). In [27] we can also see the the highly imbalanced distribution of both training and test sets.

The experiments have been carried out using our own Deep Learning framework, due to the novelty of our proposed method and the use of C++ third party libraries (*PCL*).

4.1 ModelNet10

This experiment has been carried out using a *Nurbs layer* with radius search of 0.10 (after cloud normalization) and 2 internal Nurbs, initialized randomly. In the voxelization layer, we have chosen a resolution of (1,30,30) for R, ϕ and θ

respectively and a max pooling activation strategy. The fully connected layer consists of an input layer of 1800 neurons (1x30x30x2), an internal layer of 1800 neurons with a sigmoid activation and a final softmax layer with 10 outputs. The learning rate has been set to 0.01, adaptive, and the batch size is 10.

Actual \ Predicted	bathtub	bed	chair	desk	dresser	monitor	nightstand	sofa	table	toilet
bathtub	80	12	0	0	0	0	0	6	2	0
bed	0	98	1	0	0	0	0	0	1	0
chair	0	2	96	0	0	0	0	0	2	0
desk	0	0	1	77	2	1	1	8	9	0
dresser	0	0	0	1	72	4	23	0	0	0
monitor	0	0	2	0	1	93	1	1	2	0
nightstand	0	0	0	2	21	1	71	0	5	0
sofa	0	0	0	1	1	0	1	97	0	0
table	0	1	0	17	0	0	0	0	82	0
toilet	0	1	4	0	0	0	1	0	0	94

Figure 4: Confusion matrix of the classification test results achieved by *Nurbsnet* after 50 training iterations using the *ModelNet10* dataset with 2 Nurbs. The values shown in the table are expressed as percentages. It is remarkable that there is some confusion between the pair of classes desk-table and dresser-nightstand, which are very similar.

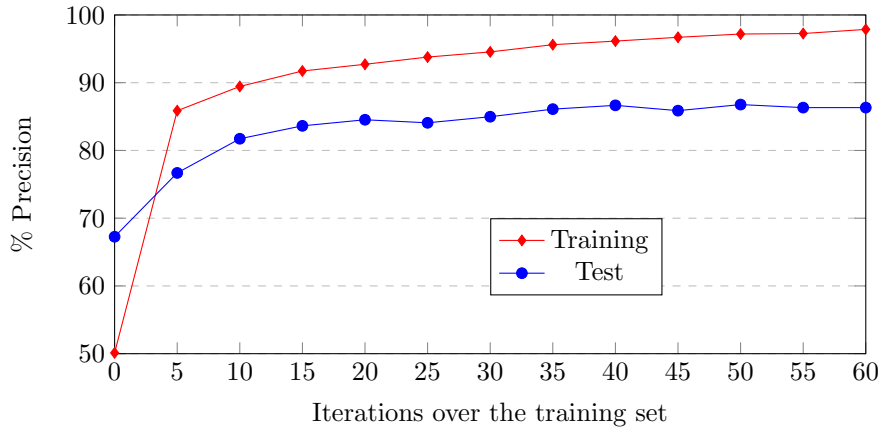

Figure 5: Training and test precision achieved by *Nurbsnet* after 60 training iterations using the *ModelNet10* dataset with 2 internal Nurbs

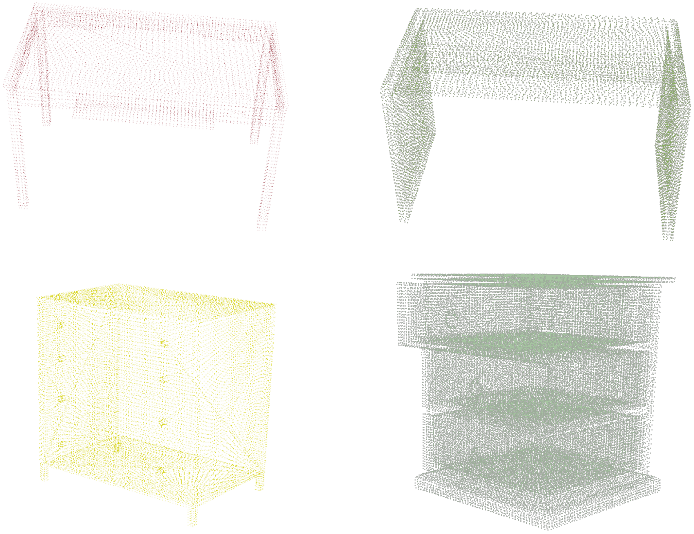

Figure 6: Pairs of confused objects. From left to right and top to down: desk classified as table, table classified as desk, dresser classified as nightstand and nightstand classified as dresser.

The final classification precision for test set is 86.77%. Despite the fact that this is a good result, this score is much more interesting if we analyse the confusion matrix provided in Figure 4. We can see that similar objects such as the pairs desk-table and dresser-nightstand are getting confused, but this kind of confusion is typical even for humans, because the difference is more semantic (the use that we make of these objects) than on their shapes. In Figure 6 we see an example of these point cloud pairs. If we count the misclassification between these pairs as a hit, the real precision would be 94.62%, which is similar to state-of-the-art methods, as shown in the table presented in [15], with much less iterations over the entire training dataset than another methods based on convolutions and only 6480000 parameters, which yields to a minimum memory consumption of 200 MB.

4.2 ModelNet40

This experiment has been carried out using a *Nurbs layer* with radius search of 0.10 (after cloud normalization) and 2 internal Nurbs, initialized randomly. In the voxelization layer, we have chosen a resolution of (1,30,30) for R, ϕ and θ respectively and a max pooling activation strategy. The fully connected layer consists of an input layer of 1800 neurons (1x30x30x2), an internal layer of 1800 neurons with a sigmoid activation and a final softmax layer with 40 outputs. The learning rate has been set to 0.01, adaptive, and the batch size is 10.

The final classification precision for test set is 75.13%. Despite the fact that

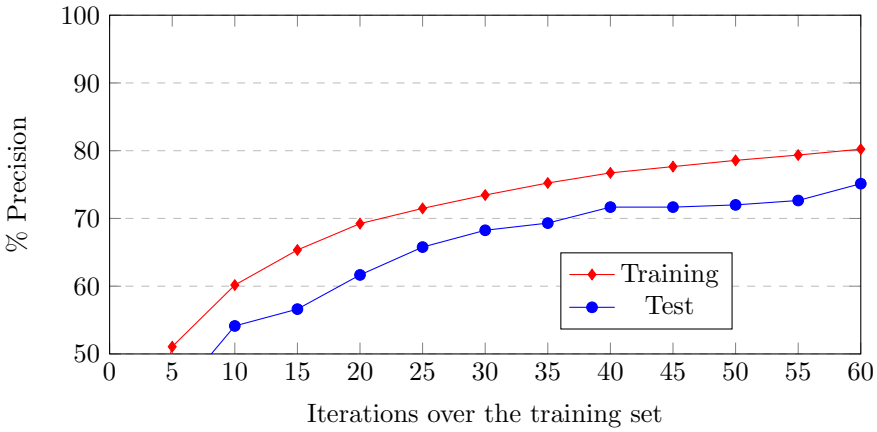

Figure 7: Training and test precision achieved by *Nurbsnet* after 60 training iterations using the *ModelNet40* dataset with 2 internal Nurbs

this is not a bad result at all for 40 classes, this score is much more interesting if we analyse the confusion matrix provided in Figure 9. We can see that there are also new similar pairs like cup-vase, flower pot-vase, flower pot-plant and curtain-door that are quite similar in many cases in the dataset and are confused. In Figure 8 we see an example of these point cloud pairs. If we count the misclassification between these pairs as a hit, the real precision would be 79.28%, which is about some of the state-of-the-art methods, with much less iterations over the entire training dataset and only 6480000 parameters, which yields to a minimum memory consumption of 200 MB, much lighter than other methods presented in [15].

5 Conclusions and Future works

In this paper we proposed *NurbsNet*, a new approach for tridimensional object recognition based on the local similarities of the objects with a set of internal Nurbs surfaces. It provides a fast convergence method, with few parameters and memory consumption, and achieves very good precision results with just a few iterations over the entire dataset.

Although the current results do not outperform the state-of-the-art methods, we consider that this completely new approach has a lot of room for improvement and can inspire new different methods to handle the 3D recognition problem.

Moreover, the internal outputs of the *Nurbs layer* and *Voxelization layer* can be inspected and easily analysed, giving a first step towards the explainability of the decisions taken by the system, a hot topic at this moment.

Following on this work, we are planning to add more *Nurbs layers* with different values of radius search, even adaptive, in order to learn characteristics of different levels of complexity from the point clouds. Moreover, we will test

Figure 8: Pairs of confused objects. From left to right and top to down: cup classified as vase, vase classified as cup, flower pot classified as vase, reference of vase object, flower pot classified as plant, reference of plant object, curtain classified as door, door classified as curtain

this approach with partial views of the objects and occlusions, from real 3D sensors as the *Kinect*.

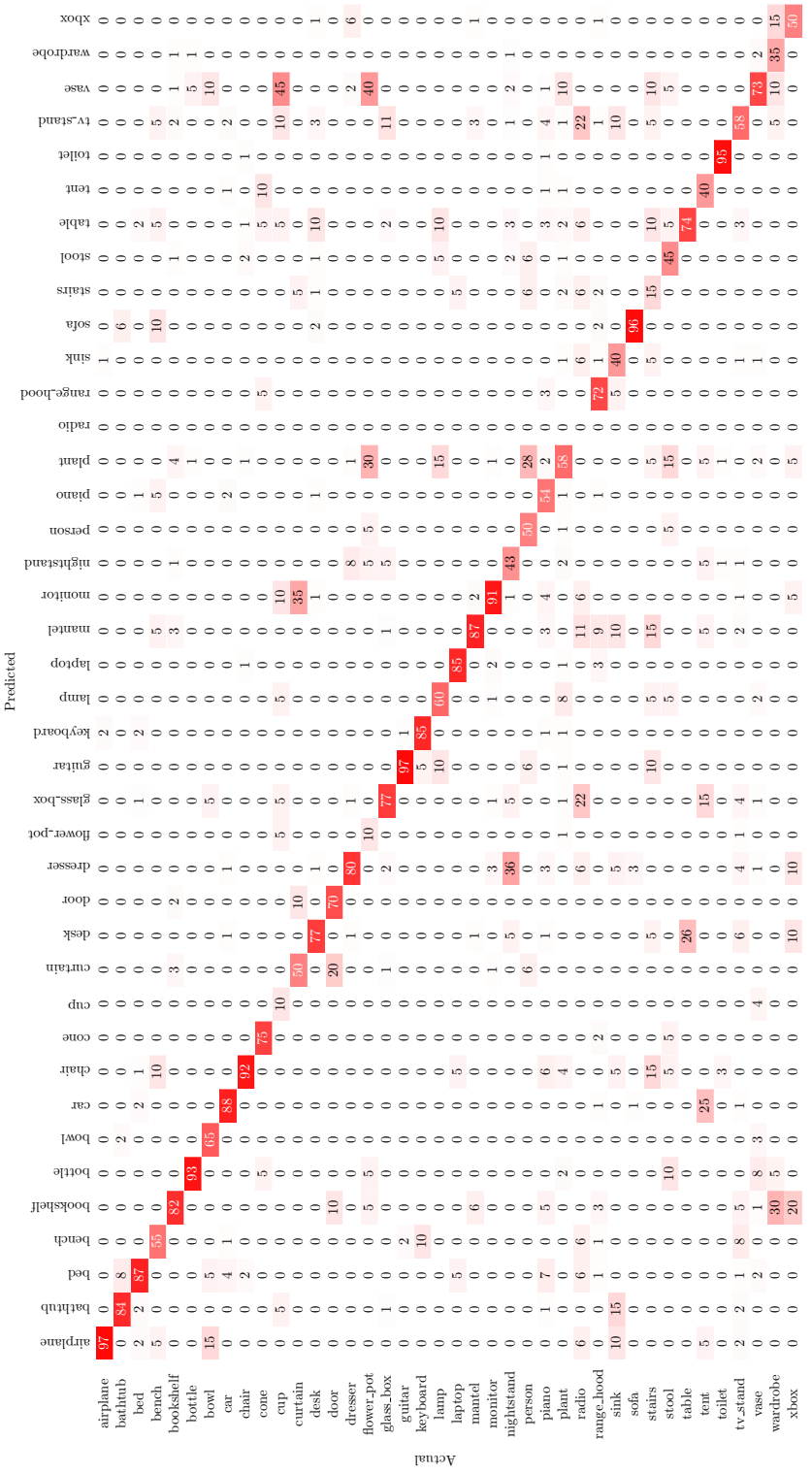

Figure 9: Confusion matrix of the classification test results achieved by *Nurbsnet* after 60 training iterations using the *ModelNet40* dataset with 2 Nurbs. The values shown in the table are expressed as percentages.

Acknowledgement

This work has been supported by the Spanish Government TIN2016-76515R Grant, supported with Feder funds. This work has also been supported by a Spanish grant for PhD studies FPU16/00887. Thanks also to Nvidia for the generous donation of a Titan Xp and a Quadro P6000.

References

[1] M. Naseer, S. Khan, and F. Porikli, "Indoor scene understanding in 2.5/3d for autonomous agents: A survey," *IEEE Access*, vol. 7, pp. 1859–1887, 2019.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *arXiv preprint arXiv:1809.02165*, 2018.

[5] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 652–660.

[6] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems*, 2017, pp. 5099–5108.

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

[8] 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)*. IEEE, 2015, pp. 922–928.

[9] A. Sharma, O. Grau, and M. Fritz, "Vconv-dae: Deep volumetric shape learning without object labels," in *European Conference on Computer Vision*. Springer, 2016, pp. 236–250.

[10] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," *arXiv preprint arXiv:1608.04236*, 2016.

[11] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," in *Advances in neural information processing systems*, 2016, pp. 82–90.

[12] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[13] E. Johns, S. Leutenegger, and A. J. Davison, "Pairwise decomposition of image sequences for active multi-view recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3813–3822.

[14] F. Gomez-Donoso, A. Garcia-Garcia, J. Garcia-Rodriguez, S. Orts-Escolano, and M. Cazorla, "Lonchanet: A sliced-based cnn architecture for real-time 3d object recognition," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 412–418.

[15] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. Modelnet benchmark leaderboard. Accessed: 2019-12-01. [Online]. Available: https://modelnet.cs.princeton.edu/

[16] Chrschn. Nurbs surface. Accessed: 2019-12-05. [Online]. Available: https://es.wikipedia.org/wiki/NURBS

[17] L. Piegl, "On nurbs: a survey," *IEEE Computer Graphics and Applications*, vol. 11, no. 1, pp. 55–71, 1991.

[18] A. Blake and M. Isard, *Active contours: the application of techniques from graphics, vision, control theory and statistics to visual tracking of shapes in motion*. Springer Science & Business Media, 1998.

[19] W. Wang, H. Pottmann, and Y. Liu, "Fitting b-spline curves to point clouds by curvature-based squared distance minimization," *ACM Transactions on Graphics (ToG)*, vol. 25, no. 2, pp. 214–238, 2006.

[20] L. Piegl and W. Tiller, *The NURBS book*. Springer Science & Business Media, 1997.

[21] T. Moerwald. Point cloud library - trimble code sprint final report. Accessed: 2019-12-01. [Online]. Available: https://modelnet.cs.princeton.edu/

[22] K. M. Liang, M. Rajeswari, and B. E. Khoo, "Nurbs: A new shape descriptor for shape-based image retrieval," Technical report, University Science Malaysia, Tech. Rep., 2003.

[23] ——, "Similarity measure determination from nurbs-warping method," in *7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002.*, vol. 3. IEEE, 2002, pp. 1222–1227.

[24] L. A. Piegl and W. Tiller, "Computing offsets of nurbs curves and surfaces," *Computer-Aided Design*, vol. 31, no. 2, pp. 147–156, 1999.

[25] L. Pagani and P. J. Scott, "Curvature based sampling of curves and surfaces," *Computer Aided Geometric Design*, vol. 59, pp. 32–48, 2018.

[26] L. I. Smith, "A tutorial on principal components analysis," Tech. Rep., 2002.

[27] A. Garcia-Garcia, F. Gomez-Donoso, J. Garcia-Rodriguez, S. Orts-Escolano, M. Cazorla, and J. Azorin-Lopez, "Pointnet: A 3d convolutional neural network for real-time object class recognition," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 1578–1584.

[28] X. Xu and S. Todorovic, "Beam search for learning a deep convolutional neural network of 3d shapes," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 3506–3511.

[29] M. Körtgen, G.-J. Park, M. Novotni, and R. Klein, "3d shape matching with 3d shape contexts," in *The 7th central European seminar on computer graphics*, vol. 3. Budmerice, 2003, pp. 5–17.

[30] F. Tombari, S. Salti, and L. Di Stefano, "Unique shape context for 3d data description," in *Proceedings of the ACM workshop on 3D object retrieval*. ACM, 2010, pp. 57–62.