This is a previous version of the article published in Computer Vision and Image Understanding. 2017, 164: 124-134. doi:10.1016/j.cviu.2017.06.006

A Study of the Effect of Noise and Occlusion on the Accuracy of Convolutional Neural Networks applied to 3D Object Recognition

Alberto Garcia-Garcia[b,1,*], Jose Garcia-Rodriguez[b,1], Sergio Orts-Escolano[c,1], Sergiu Oprea[b,1], Francisco Gomez-Donoso[c,1], Miguel Cazorla[c,1]

[a] *3D Perception Lab, University of Alicante*
[b] *Department of Computer Technology, University of Alicante*
[c] *Department of Computer Science and Artificial Intelligence, University of Alicante*

Abstract

In this work, we carry out a study of the effect of adverse conditions, which characterize real-world scenes, on the accuracy of a Convolutional Neural Network applied to 3D object class recognition. Firstly, we discuss possible ways of representing 3D data to feed the network. In addition, we propose a set of representations to be tested. Those representations consist of a grid-like structure (fixed and adaptive) and a measure for the occupancy of each cell of the grid (binary and normalized point density). After that, we propose and implement a Convolutional Neural Network for 3D object recognition using Caffe. At last, we carry out an in-depth study of the performance of the network over a 3D CAD model dataset, the Princeton ModelNet project, synthetically simulating occlusions and noise models featured by common RGB-D sensors. The results show that the volumetric representations for 3D data play a key role on the recognition process and Convolutional Neural Network can be considerably robust to noise and occlusions if a proper representation is chosen.

Keywords: Deep Learning, 3D Object Recognition, Convolutional Neural Networks, Noise, Occlusion, Caffe

*Corresponding author
Email address: agarcia@dtic.ua.es (Alberto Garcia-Garcia)

1. Introduction

Object class recognition is still one of the main challenges for a computer to achieve a deep understanding of a scene. This line of research has continuously evolved during the last years to the point where robust, scalable, and fast systems which are being applied in many situations are starting to arise. This progress has been enabled mainly by two milestones: the usage of 3D data and the development of deep learning architectures.

On the one hand, the advent of reliable and affordable RGB-D sensors, such as the Microsoft Kinect and PrimeSense Carmine, has revolutionized the field. Those sensors, together with community efforts in terms of software like the Point Cloud Library (PCL)[1] project, democratized 3D information, which is now easy to obtain and process. In this regard, we can feed the prediction systems with a new dimension of useful information. Because of that, traditional 2D object recognition pipelines have been superseded by 3D-based ones.

On the other hand, the vast majority of object recognition pipelines were typically based on manually engineered feature descriptors. Despite the success and popularity of those methods – specially for recognition in cluttered and occluded environments – they require considerable domain expertise, engineering skills, and theoretical foundations (and even if those skills are available, those systems are far from being perfect and completely robust). In order to overcome this problem, the aim of computer vision and machine learning researchers has been to replace those hand-crafted descriptors with neural networks able to learn them automatically. This insight gave birth to Convolutional Neural Networks (CNNs), which were successfully applied to image analysis with this purpose. This deep learning architecture is designed to process data in form of arrays and it has surpassed many existing methods reaching milestones in recognition tasks – mainly due to the fact that they are easy to train and generalize far better than traditional techniques. In this regard, CNNs have become the *de facto* standard to tackle the object class recognition problem, being often applied and deployed as end-to-end systems thanks to the existing frameworks.

However, there is not a clear conclusion about their performance in real-world situations rather than in standard databases.

In this work, we propose an in-depth study of the effect of adverse conditions that characterize real-world scenarios – such as noise caused by the sensor and occlusions due to the positions of the objects in the scene – on the performance of CNNs applied to 3D object class recognition. This study will provide us insight about the behavior of those systems in real-world conditions, as well as hints on how to improve them to obtain better performance in those situations.

This paper is organized as follows. Section 2 reviews state-of-the-art methods for 3D object recognition using CNNs. Section 3 discusses possible volumetric representations for 3D data. Section 4 presents the CNN architecture that will be used for our experimentation. Section 5 describes the experimentation itself, the methodology, the dataset, and the results. At last, 6 draws conclusions and future works.

2. Related Works

In this section, we will review the literature to analyze state-of-the-art volumetric representations for 3D data and also 2.5D and 3D approaches to CNNs. Due to the successful applications of CNNs to 2D image analysis, several researchers decided to increase the dimensionality of the input by adding depth information as an additional channel to conform 2.5D CNNs.

2.1. Volumetric Representations

In this subsection, we will review the most popular and successful volumetric representations for 3D data that have been used to feed CNNs for object recognition purposes.

The first step was taken by Wu *et al.* [2], their work *3DShapeNets* was the first to apply CNNs to pure 3D representations. Their proposal (shown in Figure 1) represents 3D shapes, from captured depth maps that are later transformed into point clouds, as 3D voxel grids of size $30 \times 30 \times 30$ voxels – $24 \times 24 \times 24$

(a) Object model (b) Depth map (c) Voxel grid

Figure 1: *3DShapeNets* representation proposed by Wu *et al.* as shown in their paper [2]. An object (a) is captured from a certain point of view and a depth map is generated (b) which is in turn used to generate a point cloud that will be represented as a voxel grid (c) with empty voxels (in white, not represented), unknown voxels (in blue), and surface or occupied voxels (red).

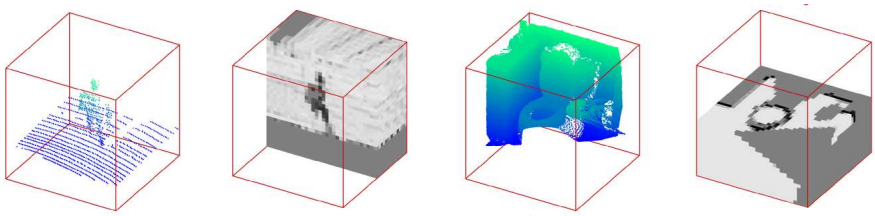
data voxels plus 3 extra ones of padding in both directions to reduce convolution artifacts – which can represent free space, occupied space (the shape itself), and unknown or occluded space depending on the point of view. Neither the grid generation process, nor the leaf size is described but the voxel grid relies on prior object segmentation.

(a) Object (b) Point cloud (c) TSDF grid

Figure 2: TSDF representation proposed by Song and Xiao as shown in their paper [3]. An object (a) is captured by a range sensor as a point cloud (b) and then a TSDF grid is generated (red indicates the voxel is in front of surfaces and blue indicates the voxel is behind the surface; the intensity of the color represents the TSDF value).

Song and Xiao [3] proposed to adopt a directional TSDF encoding which takes a depth map as input and outputs a volumetric representation. They divide a 3D space using an equally spaced voxel grid in which each cell holds a three-dimensional vector that records the shortest distance between the voxel center and the three-dimensional surface in three directions. In addition, the value is clipped by 2δ, being δ the grid size in each dimension. A $30 \times 30 \times 30$ voxels grid is fitted to a previously segmented object candidate. Figure 2 shows a graphical representation of this approach.

Maturana and Scherer [4] use occupancy grids in *VoxNet* to maintain a probabilistic estimate of the occupancy of each voxel to represent a 3D shape. This estimate is a function of the sensor data and prior knowledge. They propose three different occupancy models: binary, density, and hit. The binary and density models make use of raytracing to compute the number of hits and pass-throughs for each voxel. The former one assumes that each voxel has a binary state, occupied or unoccupied. The latter one assumes that each voxel has a continuous density, based on the probability it will block a sensor beam. The hit grid ignores the difference between unknown and free space, only considering hits; it discards information but does not require the use of raytracing so it is highly efficient in comparison with the other methods. They also propose two different grids for LIDAR and RGB-D sensor data. For the RGB-D case, they



(a) LIDAR data (b) Voxnet grid (c) RGBD data (d) Voxnet grid

Figure 3: Volumetric occupancy grid representation used by *VoxNet* as shown in their paper [4]. For LIDAR data (a) a voxel size of 0.1m^3 is used to create a $32 \times 32 \times 32$ grid (b). For RGB-D data (c), the resolution is chosen so the object occupies a subvolume of $24 \times 24 \times 24$ voxels in a $32 \times 32 \times 32$ grid (d).

use a fixed occupancy grid of $32 \times 32 \times 32$ voxels, making the object of interest – obtained by a segmentation algorithm or given by a sliding box – occupy a subvolume of $24 \times 24 \times 24$ voxels. The z axis of the grid is aligned with the direction of gravity. Figure 3 shows the occupancy grids used by *VoxNet*.

2.2. Convolutional Neural Networks

In this subsection, we will review state-of-the-art 2.5D and 3D CNNs which are applied to object recognition using 3D data.

Socher *et al.* [5] introduced a model based on a combination of CNNs and Recursive Neural Networks (RNNs) to learn features and classify RGB-D images. That model aims to learn low-level and translation invariant features with the CNN layers, those features are then given as inputs to fixed-tree RNNs to compose higher order features. Alexandre *et al.* [6] explore the possibility of transferring knowledge [7][8] between CNNs to improve accuracy and reducing training time when classifying RGB-D data. Hoeft *et al.* [9] proposed a four-stage CNN architecture, derived from the work of Schulz and Behnke [10], to semantically segment RGB-D scenes, providing the depth channel as feature maps representing components of a simplified histogram of oriented depth operator. Wang *et al.* [11] combined a CNN, to extract representative image features from RGB-D, with a Support Vector Machine (SVM) to classify objects in those images. Schwarz *et al.* [12] went one step beyond. They presented a system for object recognition and pose estimation using RGB-D images and transfer learning between a pre-trained CNN for image categorization and another CNN to classify colorized depth images. The features are then classified into instances and categories by SVMs and the pose is estimated via using another RBF kernel SVM.

In spite of the fact that those methods extend the traditional CNN, they do not employ a pure volumetric representation and therefore they do not make full use of the geometric information in the data. What is more, they do not use 3D convolutions. This is why they fall in the 2.5D CNNs category. In order to improve 2.5D CNNs, several authors proposed pure volumetric approaches

or the so called 3D CNNs. These architectures apply spatially 3D convolutions fully utilizing geometric data.

The seminal work of Wu *et al.* [2] introduced a system that supports joint object recognition and shape completion from 2.5D depth maps that are transformed into a 3D shape representation which consists of a probability distribution of binary values on a 3D voxel grid. A Convolutional Deep Belief Network (CDBN) is used to recognize categories, complete 3D shapes, and predict next best views if the recognition is uncertain. Maturana and Scherer [4] proposed a 3D CNN for landing zone detection from LIDAR data. In that work, they also introduced a volumetric representation for that data using a density occupancy grid. Later, they extended that work creating VoxNet [13] a 3D CNN architecture for real-time object classification using volumetric occupancy grids to represent point clouds.

Other remarkable works are the multi-view system by Su *et al.* [14], the panoramic network by Shi *et al.* [15], and the orientation-based voxel nets by Sedaghat *et al.* [16].

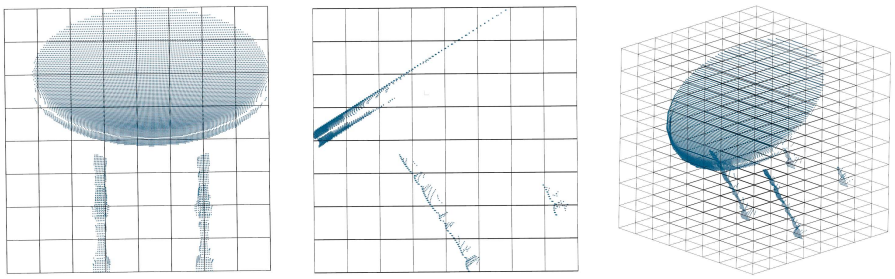
3. Volumetric Representations

As is clear from the previous sections, a volumetric representation to be fed to a 2.5D or 3DCNN must encode the 3D shape of an object as a 3D tensor of binary or real values. This is due to the fact that raw 3D data is sparse, i.e., a 3D shape is only defined on its surface, and CNNs are not engineered for this kind of data.

In this regard, our proposal for the study is twofold. First, we implemented two different ways of generating the structure of the tensor – position, grid size, and leaf size – using a fixed grid and an adaptive one. Second, we developed two possible occupancy measures for the volumetric elements of the tensor.

3.1. Tensor Generation

Providing that the input to our network consists of point clouds generated from the information provided by RGB-D sensors, we need to generate a dis-



(a) Front (b) Side (c) Perspective

Figure 4: A fixed occupancy grid ($8 \times 8 \times 8$ voxels) with 40 units leaf size and 320 units grid size in all dimensions. The grid origin is placed at the minimum x, y, and z values of the point cloud. Front (a), side (b), and perspective (c) views of the grid over a partial view of a segmented table object are shown.

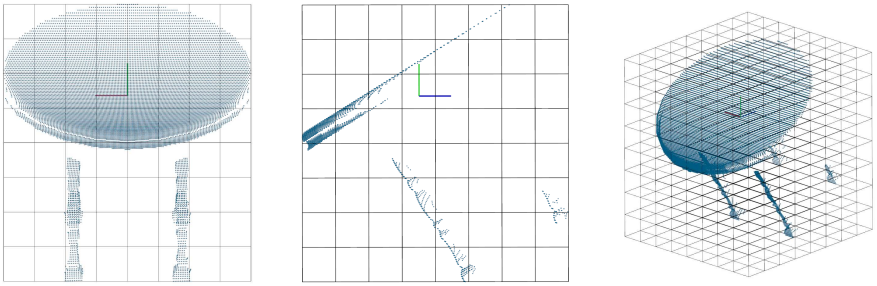
cretized representation of the unbounded 3D data to feed the network. Each cloud will be represented as a 3D tensor. For that purpose, we need to spawn a grid to subdivide the space occupied by the point clouds. Two types are proposed: one with fixed leaf and grid sizes, and another one which will adapt those sizes to fit the data.

3.1.1. Fixed

This kind of grid sets its origin at the minimum x, y, and z values of the point cloud. Then the grid is spawned, with fixed and predefined sizes for both grid and voxels. After that, the cloud is scaled up or down to fit the grid. The scale factor is computed with respect to the dimension of maximum difference between the cloud and the grid. The cloud is scaled with that factor in all axes to maintain the original ratios. As a result, a cubic grid is generated as shown in Figure 4.

3.1.2. Adaptive

The adaptive grid also sets its origin at the minimum x, y, and z values of the point cloud. Next, the grid size is adapted to the cloud dimensions. The leaf size is also computed in function of the grid size. Knowing both parameters,



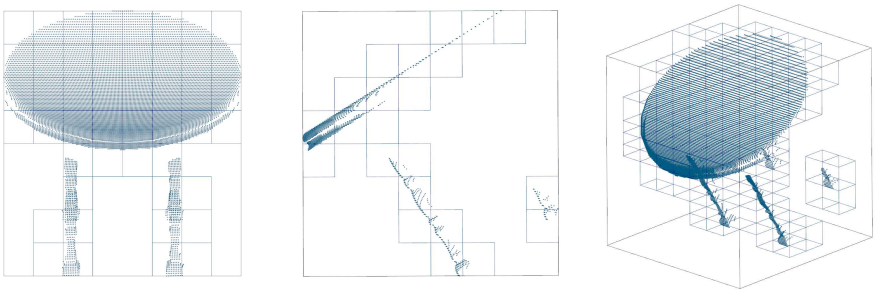
(a) Front (b) Side (c) Perspective

Figure 5: An adaptive occupancy grid ($8 \times 8 \times 8$ voxels) with adapted leaf and grid sizes in all dimensions to fit the data. The grid origin is placed at the minimum x, y, and z values of the point cloud. Front (a), side (b), and perspective (c) views of the grid over a partial view of a segmented table object are shown. Notice that the point clouds for the three views are exactly the same for this figure and Figure 4, but the grids do change. There is a noticeable difference in the front view. In Figure 4, using fixed grids, all voxels are cubic and the point cloud does not fit the grid completely (leftmost column in Figure 4a), whilst in this figure, with adaptive grids, the grid is fitted to the cloud.

the grid is spawned, fitting the point cloud data. As a result, a non-cubic grid is generated. As shown in Figure 5, all voxels have the same size, but they are not necessarily cubic.

It is important to remark that, in both cases (fixed and adaptive), the number of voxels in the grid is fixed. Figures 4 and 5 show examples for both types using $8 \times 8 \times 8$ voxels for the sake of a better visualization.

It is also important to notice that each representation serves a purpose. The fixed grid will not always fit the data perfectly so it might end up having sparse zones with no information at all (as seen in Figure 4a on the first column). However, it can be used right away for sliding box detection. On the contrary, the adaptive grid fits the data to achieve a better representation. Nonetheless, it relies on a proper segmentation of the object to spawn the grid.



(a) Front (b) Side (c) Perspective

Figure 6: Occupied voxels in an adaptive $8 \times 8 \times 8$ grid generated over a partial view point cloud. Those voxels with points inside are shown in a wireframe representation. Empty voxels are omitted. Occupied voxels must be filled with values which represent the contained shape.

3.2. Occupancy Computation

After spawning the grid to generate a discrete space, we need to determine the values for each cell or voxel of the 3D tensor. In order to do that, we must encode the geometric information of the point cloud into each occupied cell (see Figure 6). In other words, we have to summarize as a single value, the information of all points which lie inside a certain voxel. One way to do that is using occupancy measures. For that purpose, we propose two different alternatives: binary occupancy, normalized density.

3.2.1. Binary

The binary tensor is the simplest representation that can be conceived to encode the shape. Voxels will hold binary values, they will be considered occupied if at least a point lies inside, and empty otherwise. Figure 7 shows an example of this tensor.

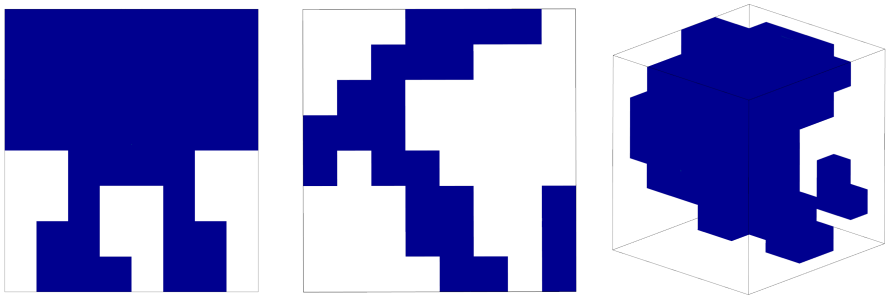
3.2.2. Normalized Density

Binary representations are simple and require low computational power. However, complex shapes may get oversimplified so useful shape information gets lost. This representation can be improved by taking into account more shape information. A possible alternative consists of computing the point density inside each voxel, i.e., counting the number of points that fall within each cell.

It is important to notice that point density directly depends on the cloud resolution which in turn depends on many factors involving the camera and the scene, e.g., it is common for RGB-D to generate denser shapes in closer surfaces. To alleviate this problem, we can normalize the density inside each voxel dividing each value by the maximum density over the whole tensor. An example of normalized density tensor is shown in Figure 8.

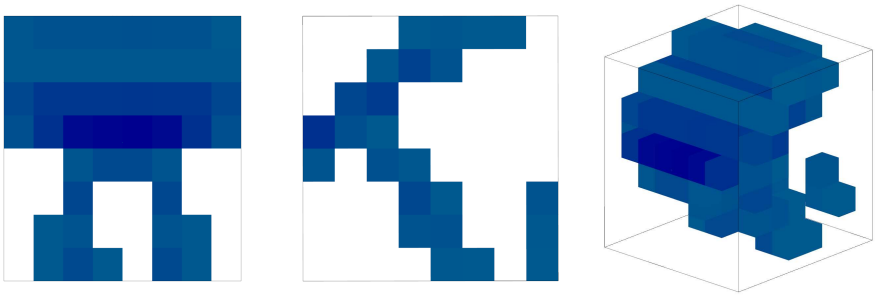
4. Convolutional Neural Network

In this section, we will describe the main layers that compose the CNN that will be used for the study. Figure 9 shows a diagram of the chosen architecture. It is highly inspired by *Voxnet* [4] and *PointNet* [17]. The network was imple-



(a) Front (b) Side (c) Perspective

Figure 7: Binary tensor computed over a point cloud of a partial view of an object (shown in Figure 6). Occupied voxels are shown in blue, empty voxels are omitted for the sake of simplicity.



(a) Front (b) Side (c) Perspective

Figure 8: Normalized density tensor over a point cloud of a partial view of an object (shown in Figure 6). Denser voxels are darker and sparse ones are shown in light blue. Empty voxels were removed for visualization purposes.

mented using Caffe. It features 2D convolutions and takes full 3D object model point clouds as input (more details about this input are provided in Section 5.1).

The input layer is a custom data layer implemented in Caffe which takes object point clouds as inputs and generates the corresponding discrete volumetric representation as discussed in the previous section.

Next, we can find a *convolution layer* or $C(m, n, d)$. This layer applies m filters of size $n \times n$ and a stride of $d \times d$ voxels. In our case, this first convolution layer learns 48 3×3 filters using a stride of 1×1 voxels. This convolution layer is followed by a Rectified Linear Unit (ReLU) activation to introduce non-linearities to the model.

After that, another convolution layer is found. In this case, it will learn 128 5×5 filters with a stride of 1×1 voxels again. This layer is also followed by a ReLU activation one.

A *pooling layer* or $P(n, d)$ takes place after those blocks. It performs a max-pooling process to summarize the input data, taking the maximum value of a fixed local spatial region of $n \times n$ which is slided across the input volume using a stride of $d \times d$ voxels. In this case, a pooling region of 2×2 voxels with the same stride was chosen.

At last, we can find an *inner product layer* or $IP(n)$. It is just a fully connected layer, a traditional neural network architecture which consists of n neurons (1024 in this case). It is followed by a ReLU activation and a *dropout layer* [18] or $DP(r)$. The function of the dropout layer is to avoid overfitting, randomly dropping connections with a probability r (0.5 in our case). In the end, another fully connected layer represents the output of the network, with as many output neurons as classes has our classification problem. Since our dataset has 10 classes (see Section 5.1) this layer has 10 neurons.

We use the term 2.5D to refer to this network due to the fact that it processes 3D data using 2D convolutions. This means that, in the end, its convolutions do not fully take into account the depth spatial dimension of the input as if we were using pure 3D convolution filters. It is intuitive to think that a 3D CNN would yield better results due to that extra spatial dimension. However, a 3D CNN has some disadvantages that made us consider using a 2.5D CNN instead for the experimentation: (1) higher computational cost, (2) memory footprint is also much higher, (3) more parameters thus harder training. For those reasons, the main body of the experiments were carried out using the 2.5D approach.

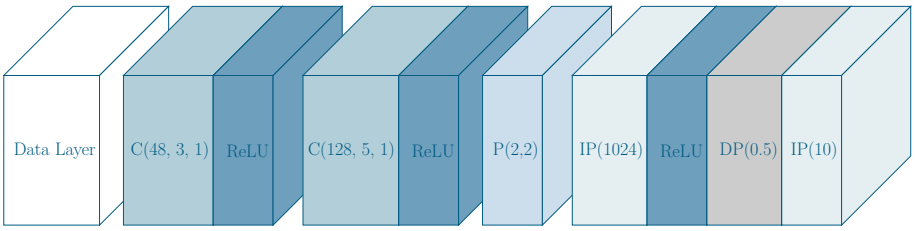


Figure 9: 2.5D Convolutional Neural Network architecture used for the experiments. This network is an extension of the one presented in PointNet [17]. It consists of a convolution layer – 48 filters, 3×3 filter with stride 1 –, a ReLU activation, another convolution layer – 128 filters, 5×5 filters with stride 1 –, followed by a ReLU activation, a pooling layer – 2×2 max. pooling with stride 2 –, a fully connected or inner product layer with 1024 neurons and ReLU activation, a dropout layer – 0.5 rate –, and an inner product layer with 10 neurons as output. The network accepts 3D tensors as input.

5. Experimentation

In order to assess the performance of the proposed model-based CNN we carried out an extensive experimentation to determine the accuracy of the model and its robustness against occlusions and noise – situations that often occur in real-world scenes. For that purpose we started using the normalized density grids since they offer a good balance between efficiency and representation. We also investigated the effect of both fixed and adaptive grids using different sizes. Further experimentation was performed to compare the normalized density grids with the binary ones. We also carried out a brief experiment using a 3D CNN to compare its performance with the 2.5D counterpart.

The networks were trained for a maximum of 5000 iterations – weights were snapshotted every 100 iterations so in the end we selected the best sets of them as if we were early stopping – using Adadelta as optimizer with $\delta = 1 \cdot 10^{-8}$. The regularization term or weight decay in Caffe was set to $5 \cdot 10^{-3}$. A batch size of 32 training samples was chosen.

Results were obtained using the following test setup: Intel Core i7-5820K with 32 GiB of Kingston HyperX 2666MHz and CL13 DDR4 RAM on an Asus X99-A motherboard (Intel X99 chipset). Additionally, the system included an NVIDIA Tesla K40c GPU used for training and inference. The framework of choice was Caffe RC2 running on Ubuntu 14.04.02. It was compiled using CMake 2.8.7, g++ 4.8.2, CUDA 7.5, and cuDNN v3.

5.1. Dataset

Deep neural network architectures are usually composed by many layers which in turn mean many weights to be learned. Because of that, there is a strong need of large-scale datasets to train those networks in order to avoid overfitting the model to the input data. Nowadays, large-scale databases of real-world 3D objects are scarce, some of them do not have that high number of objects [19][20][21], or were incomplete by the time this work was performed [22]. A possible workaround to this problem consists of using Computer Aided

Design (CAD) model databases – which are virtually unlimited – and processing those models to simulate real-world data.

The *Princeton ModelNet* project is one of the most popular large-scale 3D object dataset. Its goal, as their authors state, is to provide researchers with a comprehensive clean collection of 3D CAD models for objects, which were obtained via online search engines. Employees from the Amazon Mechanical Turk (AMT) service were hired to classify over 150 000 models into 662 different categories.

At the moment, there are two versions of this dataset publicly available for download[1]: *ModelNet-10* and *ModelNet-40*. Those are subsets of the original dataset which only provide the 10 and 40 most popular object categories respectively. These subsets are specially clean versions of the complete dataset.

On the one hand, ModelNet-10 is composed of a collection of over 5000 models classified into 10 categories and divided into training and test splits. In addition, the orientation of all CAD models of the dataset was manually aligned. On the other hand, ModelNet-40 features over 9800 models classified into 40 categories, also including training and test sets. However, the orientations of its models are not aligned as they are in ModelNet-10.

For this work, we will use of the ModelNet-10 subset, which contains a reasonable amount of models for both training and validation, mainly because this dataset was completely cleaned and the orientation of the models were manually aligned. Figure 10 shows some model examples from ModelNet-10.

The CAD models are provided in Object File Format (OFF). Firstly, we converted all OFF models into Polygon File Format (PLY) to ease the usage of the dataset with the PCL. As we already mentioned, the input for *PointNet* are point clouds, but the dataset provides CAD models specifying vertices and faces. In this regard, we converted the PLY models into Point Cloud Data (PCD) clouds by raytracing them. A 3D sphere is tessellated and a virtual camera is placed in each vertex of that truncated icosahedron – pointing to the origin

[1] `http://modelnet.cs.princeton.edu/`

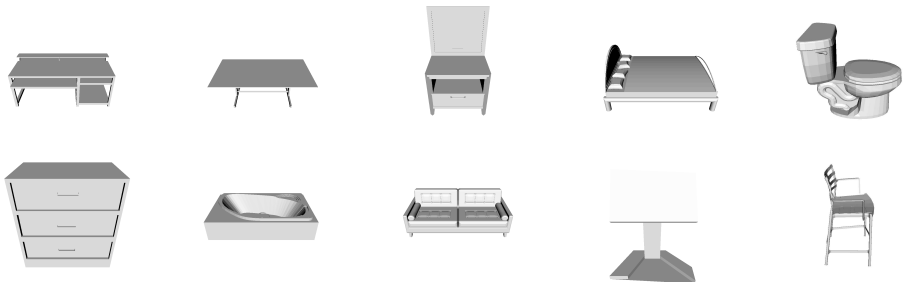


Figure 10: ModelNet10 samples.

of the model – then multiple snapshots are rendered using raytracing and the z-buffer data, which contains the depth information, is used to generate point clouds from each point of view. After all points of view have been processed, the point clouds are merged. A voxel grid filter is applied to downsample the clouds after the raytracing operation.

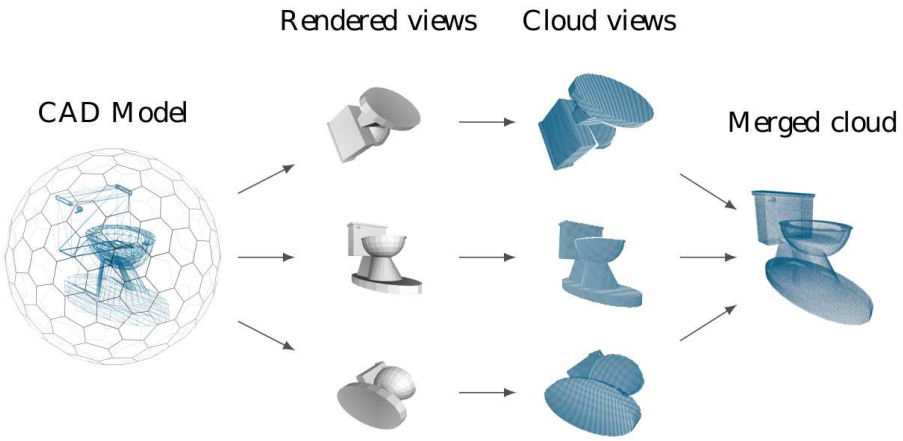
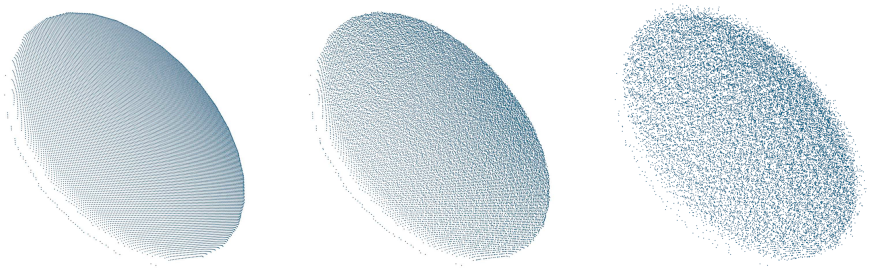


Figure 11: From CAD models to point clouds. The object is placed in the center of a tessellated sphere, views are rendered placing a virtual camera in each vertex of the icosahedron, the z-buffer data of those views is used to generate point clouds. At last, the point clouds are transformed and merged.



(a) $\sigma = 0$ (b) $\sigma = 0.1$ (c) $\sigma = 1$

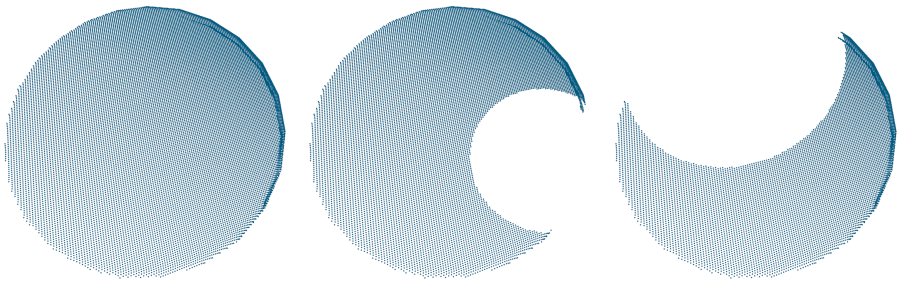
Figure 12: Different levels of noise ($\sigma = 0$ (a), $\sigma = 0.1$ (b), and $\sigma = 1$ (c)) applied to the z-axis of every point of a table partial view.

5.2. Noise Simulation

The partial views generated using the previously described process are not a good simulation of the result that we would obtain by using a low-cost RGB-D sensor. Those systems are noisy, so the point clouds produced by them are not a perfect representation of the real-world objects.

In order to properly simulate the behavior of a sensor, a model is needed. In our case, we are dealing with low-cost RGB-D sensors such as Microsoft Kinect and Primesense Carmine. A complete noise model for those sensors, specifically for the Kinect device, must take into account occlusion boundaries due to distance between the Infrarred (IR) projector and the IR camera, 8-bit quantization, 9×9 pixel correlation window smoothing, and z-axis or depth Gaussian noise [23].

We will make use of a simplification of this model, only taking into account the Gaussian noise since it is the most significant one for the generated partial views. In this regard, the synthetic views are augmented by adding Gaussian noise to the z dimension of the point clouds with mean $\mu = 0$ and different values for the standard deviation σ to quantify the noise magnitude. Figure 12 shows the effect of this noise over a synthetic partial view of one object of the dataset.



(a) $\psi = 0\%$ (b) $\psi = 25\%$ (c) $\psi = 50\%$

Figure 13: Different levels of occlusion ($\psi = 0\%$ (a), $\psi = 25\%$ (b), and $\psi = 50\%$ (c)) applied randomly to a table partial view.

5.3. Occlusion Simulation

In addition to modelling the sensor to improve our synthetic data, it is important to also take the environment into account. In a real-world scenario, objects are not usually perfectly isolated and easily segmented; in fact, it is common for them to be occluded by other elements of the scene.

The occlusion simulation process consists of picking a random point of the cloud with a uniform probability distribution. Then, a number of closest neighbors to that point are picked. At last, both the neighbors and the point are considered occluded surface and removed from the point cloud. The number of neighbors to pick depends on the amount of occlusion ψ we want to simulate. For instance, for an occlusion $\psi = 25\%$ we will remove neighbors until the rest of the cloud contains a 75% of the original amount of points, i.e., we will remove a 25% of the original cloud. Figure 13 shows the effect of the random occlusion process with different occlusion factors ψ over a synthetic partial view of a table object of the dataset.

It is important to notice the randomness of the occlusion process. This means that even with a high ψ it is possible not to remove any important surface information and vice versa. In other words, it is possible for some objects to remove a 50% of their points and still be recognizable because the removed region was not significant at all, e.g., a completely flat surface. However it is

possible to render an object unrecognizable by removing a small portion of its points if the randomly picked surface is significant for its geometry. This remark is specially important when testing the robustness of the system. In order to guarantee that an appropriate measure of the robustness against missing information is obtained, a significant amount of testing sets must be generated and their results averaged so that it is highly probable to test against objects which have been occluded all over their surface across the whole testing set.

5.4. Results

After describing the experimentation setup, the dataset that was used to train and test the networks, and the ways of simulating noise and occlusion for the test sets, we will present and discuss the results of the experiments. Firstly, the normalized density tensor results – using the 2.5D CNN – will be presented. After that, we will proceed with the binary tensor ones. Furthermore, we will report the experiments which produced the best results with a pure 3D CNN with fully 3D convolutions. **At last, we will perform a comparison with the state of the art.**

5.4.1. Density Tensor

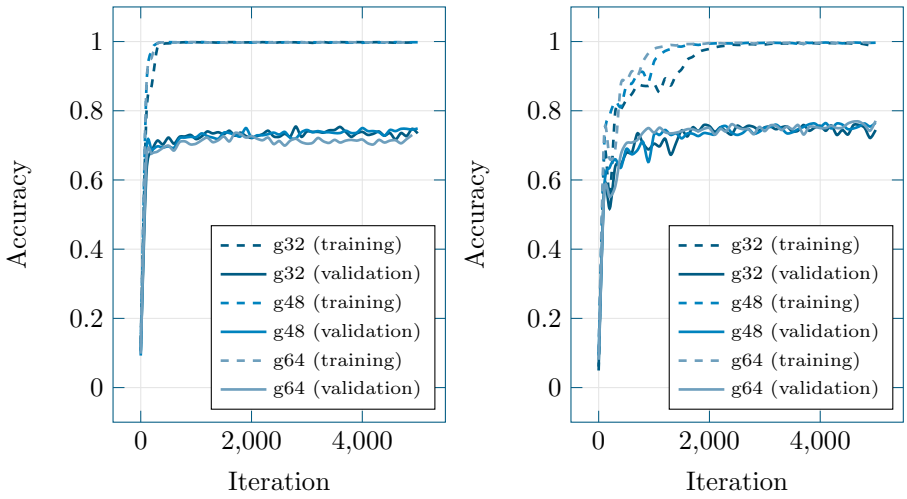
Figure 14 shows the accuracy results of the network for both grid types and increasing sizes. The peak accuracies for the fixed grids are ≈ 0.75, ≈ 0.76, and ≈ 0.73 for sizes 32, 48, and 64 respectively. In the case of the adaptive one, the peak accuracies are ≈ 0.77, ≈ 0.78, and ≈ 0.79 for the sizes 32, 48, and 64 respectively.

Taking those facts into account, we can extract two conclusions. First, the adaptive grid is able to achieve a slightly better peak accuracy in all cases; however, the fixed grid takes less iterations to reach accuracy values close to the peak in all cases. Second, there is no significant difference in using a bigger grid size of 64 voxels instead of a smaller one of 32.

The most important fact that can be observed in the aforementioned figures is that there is a considerable gap between training and validation accuracy in

all situations. As we can observe, all networks reach maximum accuracy for the training set whilst the validation one hits a glass ceiling at approximately 0.80. We hypothesize that the network suffers overfitting even when we thoroughly applied measures to avoid that. The most probable cause for that problem is the reduced number of training examples. In the case of ModelNet10 the training set consists of only 3991 models. Considering the complexity of the CNN, it is reasonable to think that the lack of a richer training set is causing overfitting.

Concerning the robustness against occlusion, we took the best networks after training and tested them using the same validation sets as before but introducing occlusions in them (up to a 30%). Figure 15 shows the accuracy of both grid types with different sizes as the amount of occlusion in the validation model increases. As we can observe, occlusion has a significant and negative impact on the fixed grid – bigger grid sizes are less affected – going down from ≈ 0.75 accuracy to $0.40 - 0.50$ approximately in the worst and best case respectively when a 30% of the model is occluded. On the contrary, the adaptive grid does



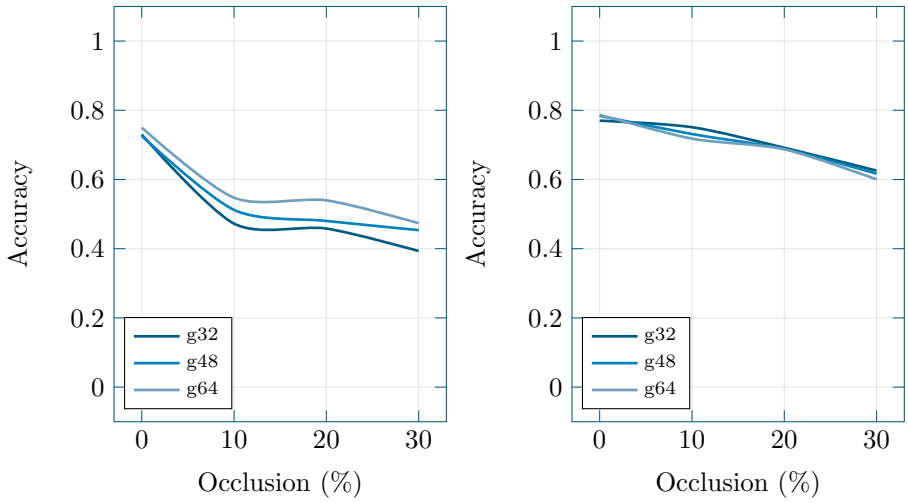
(a) Fixed Grid (b) Adaptive Grid

Figure 14: Evolution of training and validation accuracy of the model-based CNN using both fixed (a) and adaptive (b) normalized density grids. Different grid sizes (32, 48, and 64) were tested.

not suffer that much – it goes down from ≈ 0.78 to ≈ 0.60 in the worst case – and there is no significant difference between grid sizes. In conclusion, the adaptive grid is considerably more robust to occlusion than the fixed one.

Regarding the resilience to noise, we also tested the best networks obtained from the aforementioned training process using validation sets with different levels of noise (ranging from $\sigma = 1 \cdot 10^{-2}$ to $\sigma = 1 \cdot 10^{1}$). Figure 16b shows the results of those experiments. It can be observed that adding noise has a significant impact on the fixed grid, even small quantities, reducing the accuracy from ≈ 0.75 to ≈ 0.60, ≈ 0.4, and ≈ 0.2 for $\sigma = 1{\cdot}10^{-1}, \sigma = 1{\cdot}10^{0}$, and $\sigma = 1{\cdot}10^{1}$ respectively. On the other hand, the adaptive one shows remarkable robustness against low levels of noise (up to $\sigma = 1 \cdot 10^{-1}$), barely diminishing its accuracy.

In the end, both grids suffer huge penalties in accuracy when noise levels higher than $\sigma = 1 \cdot 10^{-}1$ are introduced, being the adaptive one less affected. The grid size has little to no effect in both cases, only in the fixed grid bigger sizes are slightly more robust when intermediate to high levels of noise are



(a) Fixed Grid (b) Adaptive Grid

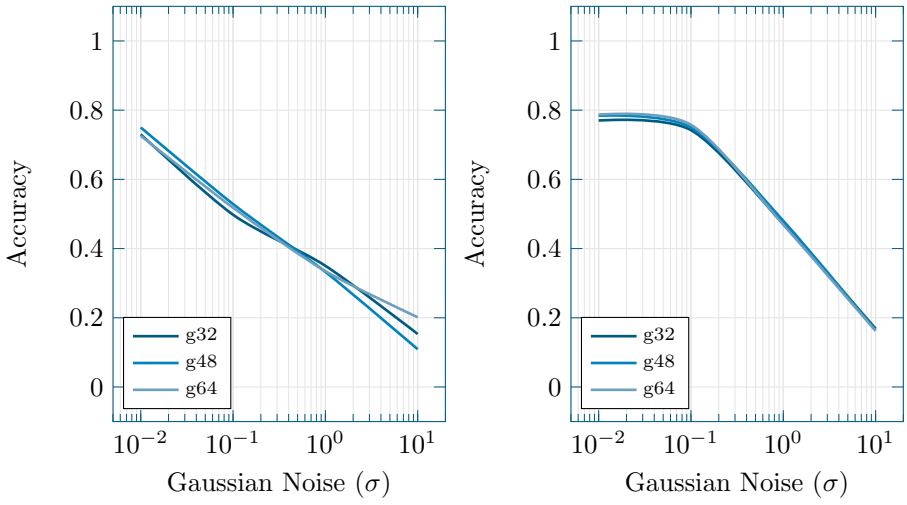
Figure 15: Evolution of validation accuracy of the model-based CNN using both fixed (a) and adaptive (b) normalized density grids as the amount of occlusion in the validation models increases from 0% to 30%. Three grid sizes were tested (32, 48, and 64).

introduced. In conclusion, the adaptive grid is significantly more resilient to low levels of noise, and slightly outperforms the fixed one when dealing with intermediate to high ones.

5.4.2. Binary Tensor

After testing the performance of the normalized density grid, we also trained and assessed the accuracy of the binary one in the same scenarios. This test intended to show whether there is any gain in using representations which include more information about the shape – at a small penalty to execution time.

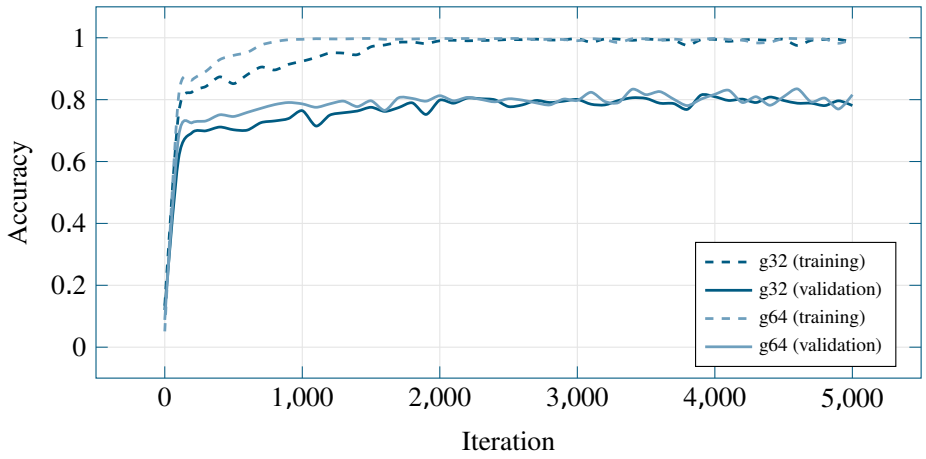
For this experimentation we picked the best performer in the previous sections: the adaptive grid. We also discarded the intermediate size (48 voxels) since there was no significant difference between it and the others. Figure 17a shows the accuracy results of the network trained using binary grids. As we can observe, there is no significant difference between grid sizes neither. However,



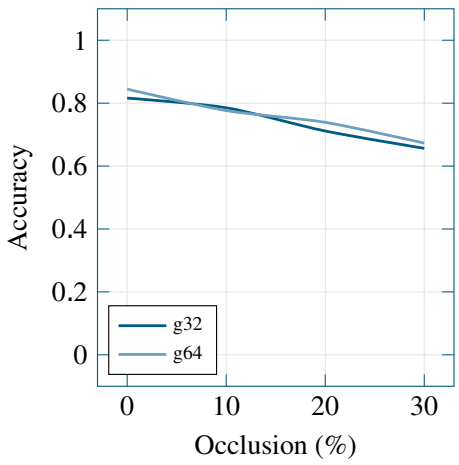
(a) Fixed Grid (b) Adaptive Grid

Figure 16: Evolution of validation accuracy of the model-based CNN using both fixed (a) and adaptive (b) normalized density grids as the standard deviation of the Gaussian noise introduced in the z-axis of the views increases from 0.001 to 10. The common grid sizes were tested (32, 48, and 64).

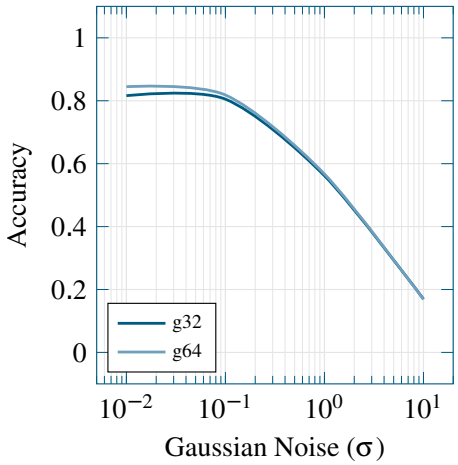
using this representation we achieved a peak accuracy of approximately 0.85, using 64 voxels grids, which is better to some extent than the normalized density one shown in Figure 14.



(a) Training



(b) Occlusion



(c) Noise

Figure 17: Evolution of training and validation accuracy of the model-based CNN using adaptive binary grids (a). Evolution of validation accuracy for the best network weights after training as the amount of occlusion in the validation set increases (b) and different levels of noise are introduced (c).

Occlusion and noise tolerance (shown in Figures 17b and 17c respectively) is mostly similar to the robustness shown by the normalized density adaptive grid (see Figures 15b and 16b) except from a small offset caused by the higher accuracy of the binary grid network.

In conclusion, the less-is-better effect applies in this situation and turns out that the simplification introduced by the binary representation helps the network during the learning process. It is pending to check if this statement is still valid if the validation accuracy is not bounded by network overfitting.

5.4.3. 3D CNN

At last, we tested the best configuration – binary adaptive grids – with a 3D CNN architecture with pure 3D convolutions. We kept the same architecture we introduced in Section 4, but extended its convolution and pooling layers to three dimensions. We then trained the network using adaptive binary grids as the volumetric representation of choice and monitored validation and training errors. Due to memory limitations on the GPU we could only experiment with grids of $32 \times 32 \times 32$ voxels.

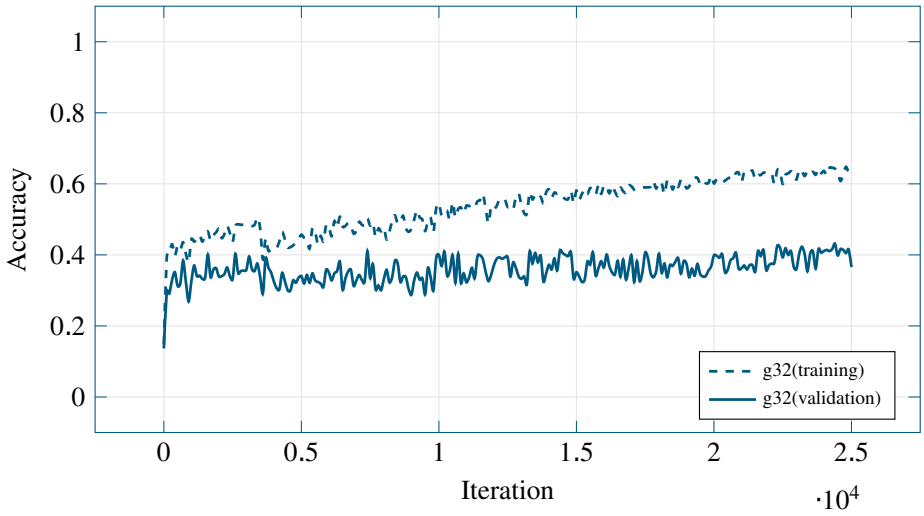


Figure 18: Evolution of training and validation accuracy of the 3D CNN using adaptive binary grids with size $32 \times 32 \times 32$.

Figure 18 shows the results of this experiment. As we can observe, we trained the network for five times more iterations than before and even then we couldn't achieve a proper convergence. The training set accuracy kept increasing slowly up to approximately 0.65 whilst the validation one got stuck around 0.40 for the whole experiment.

In conclusion, porting the 2.5D network directly to 3D just by extending its convolution and pooling layers to slide along the depth axis did not produce good results using the same dataset and setup that produced a significantly good outcome with the 2.5D architecture. We hypothesize various causes for this problem.

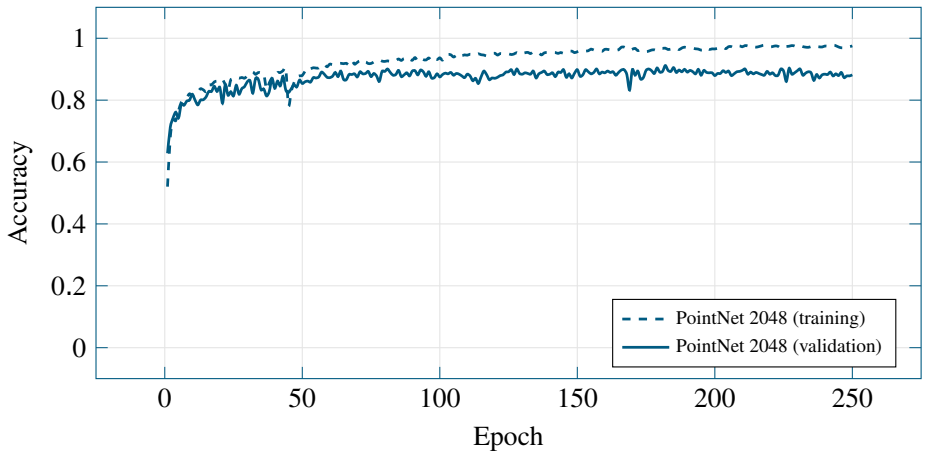
On the one hand, the data representation might not be adequate for such fine-grained convolutions. It is presumable that bigger grids, e.g., $64 \times 64 \times 64$, would yield better results. However, given the size of the model, they could not be tested in the available GPU.

On the other hand, the complexity of the network increased considerably after including that extra dimension in convolution and pooling layers. This means that the number of parameters of the network gets increased significantly, making it harder to train with so few samples due to overfitting. This hypothesis is backed up by the fact that training accuracy kept increasing slowly while validation one got stuck. This would eventually lead to a perfect fit on the training set but low accuracy on the validation split.

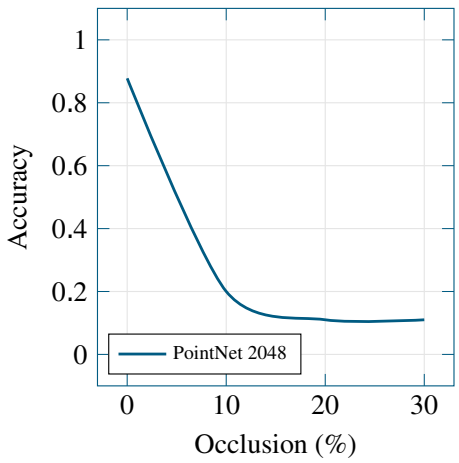
5.4.4. Comparison with State-of-the-art Methods

In order to assess the validity of our proposal and conclusions, we analyzed the state of the art to find other methods which deal with 3D point clouds directly. We found out that the best method, in ModelNet-40, which provided an implementation that could be reproduced was PointNet [24]. PointNet's approach is particularly interesting since they do not rely on any traditional CNN-style architecture. Instead, their deep network is mainly composed by Multi-Layer Perceptrons (MLPs).

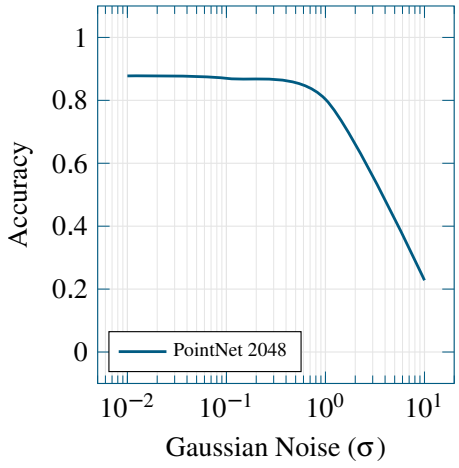
In order to test PointNet under fair conditions, we regenerated all training and testing data using their pipeline for mesh sampling (with 2048 **points and leaf size** 0.005**) and unit sphere normalization. After**



(a) Training



(b) Occlusion



(c) Noise

Figure 19: **Evolution of training and validation accuracy of PointNet (a). Evolution of validation accuracy for the best network weights after training as the amount of occlusion in the validation set increases (b) and different levels of noise are introduced (c).**

generating ModelNet-10 this way, we applied noise and occlusions as described before. Then we trained PointNet using ModelNet-10's training and validation partitions. Finally, we tested that trained model for occlusion and noise resilience.

Figure 19 shows the results of this experiments. The network was trained for 250 epochs using a batch size of 32. The decay rate was set to 0.7, the decay step to 200000, and the learning rate to 0.001. In the end, the best set of weights achieved a validation accuracy of approximately 0.90 while the training accuracy kept increasing until 0.97, thus showing clear signs of overfitting. The most remarkable fact to notice is the extremely negative impact that occlusion has in this architecture (see Figure 19b). As we can observe, accuracy drops to 0.20 with 10% occlusion and ends at 0.11 with 30% of points occluded. On the one hand, PointNet is clearly outperformed by our previous approaches by a large margin in any occlusion level. On the other hand, PointNet exhibits a much stabler behavior when dealing with noise, being able to keep accuracy without any significant drop until $\sigma = 1 \cdot 10^1$. With a relatively high level of noise such as $\sigma = 1 \cdot 10^0$, accuracy is still way over 0.80; however, when noise gets to $\sigma = 1 \cdot 10^1$ it drops significantly to 0.23.

5.5. Discussion

To sum up, we determined that the adaptive grid slightly outperforms the fixed one in normal conditions. The same happens with the grid size, obtaining marginally better results with bigger sizes. However, when it comes down to noise and occlusion robustness, the adaptive grid exceeds the accuracy of the fixed grid by a large margin for low levels of occlusion and noise, whilst for intermediate and high levels the impact on both grids is somewhat similar. In other words, the adaptive grid is better than the fixed one and it is preferable to use a bigger grid size if the performance impact can be afforded.

It is important to remark that the binary occupancy measure performed

better than the normalized density one, both using adaptive grids, while maintaining similar resilience against noise and occlusions. The best network trained with normalized density grids reached a peak accuracy of approximately 0.79 while the best binary one achieved approximately a 0.85 accuracy on the validation set.

Another remarkable fact was that all networks exhibited a considerable amount of overfitting, i.e., training accuracy was almost perfect whilst validation was far away from it by a considerable margin. We hypothesize that this was due to the fact that the dataset has few training examples considering the complexity of the network. Besides, we also inspected the confusion matrix shown in Table 1 to gain insight about the behavior of our network. As we can observe, there are many misclassified samples of classes that are similar. If we take a closer look at some of the misclassified samples (see Figures 20, 21, and 22) it is reasonable to think that the network is not able to classify them properly because they are extremely similar. In this regard, the dataset must be augmented introducing noise, translations, rotations, and variations of the models to avoid overfitting and learn better those models that can be easily misclassified.

Desk	Table	Nstand	Bed	Toil.	Dresser	Bath.	Sofa	Moni.	Chair
52	9	1	4	0	5	1	5	0	9
25	69	0	1	0	0	0	0	1	4
1	2	60	1	4	8	0	0	2	8
4	0	0	80	0	0	3	11	1	1
1	0	3	1	84	0	1	3	2	5
3	0	14	0	0	61	0	1	6	1
0	1	0	3	0	0	34	8	3	1
1	0	1	4	1	2	0	88	1	2
1	1	1	1	0	5	1	1	87	2
1	2	1	2	1	1	0	1	1	90

Table 1: Confusion matrix of the validation results achieved by the best set of weights for the 2.5D CNN with binary adaptive grids with a grid size of 64 voxels. Darker cells indicate more predictions while lighter ones indicate less.

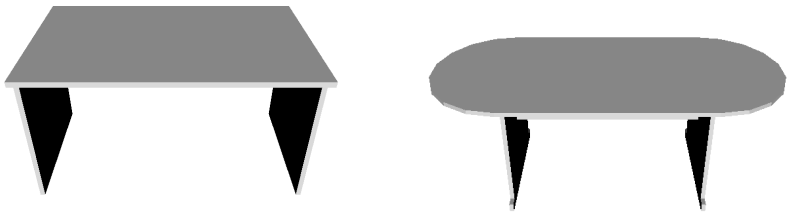


Figure 20: A desk class sample together with a table class one.

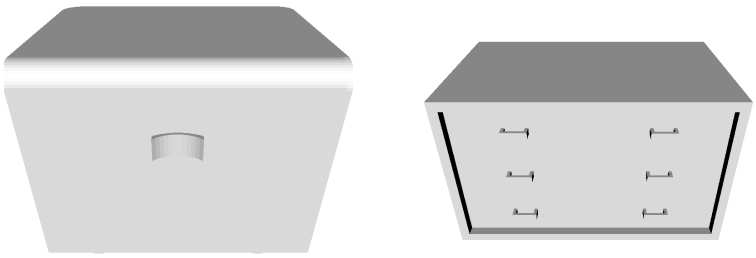


Figure 21: A night stand class sample together with a dresser one.

Figure 22: A sofa class sample together with a bed class one.

In addition, we trained the 3D CNN as before using adaptive binary grids. The results were negative in the sense that overfitting was accentuated due to the increased complexity of the network.

Grid Size	Fixed Density 2.5D	Adaptive Density 2.5D	Adaptive Binary 2.5D	Adaptive Binary 3D
$32 \times 32 \times 32$	0.75	0.77	0.80	0.43
$48 \times 48 \times 48$	0.76	0.78	N/A	N/A
$64 \times 64 \times 64$	0.73	0.79	0.85	N/A

Table 2: Summary of the experimentation results.

Occlusion (%)	PointNet 2048	2.5D Adaptive Binary (64)	Noise (σ)	PointNet 2048	2.5D Adaptive Binary (64)
0	0.90	0.85	10^{-2}	0.88	0.84
10	0.20	0.78	10^{-1}	0.87	0.82
20	0.11	0.74	10^{0}	0.80	0.57
30	0.11	0.67	10^{1}	0.23	0.18

Table 3: Summary of the comparison of our best approach (2.5D Adaptive Binary with $64 \times 64 \times 64$ grids) versus PointNet 2048.

At last, we compared our approaches with the best state-of-the-art method in the challenge which provided an implementation and enough information to reproduce their results: PointNet. That architecture, based on MLPs, achieved a slightly better base accuracy on ModelNet-10's test set (0.90 **against** 0.85**). It also showed a remarkable robustness against noise (better for high levels of noise (**$\sigma = 10^{0}$**) than our best approach but on par for low ones (**$\sigma = 10^{-2}$ **and** $\sigma = 10^{-1}$**). However, it is extremely sensitive to occlusions in comparison with our approaches (while our best adaptive binary grid keeps accuracy above** 0.65 **even for** 30% **occlusion, PointNet's accuracy drops below** 0.20 **even with just** 10% **occlusion).**

A summary of the experimentation results with the top accuracies for each configuration is shown in Table 2. In addition, Table 3 shows a summary of our best configuration against PointNet.

6. Conclusion

In this paper we have presented a study of the effect of adverse conditions on the accuracy of CNNs trained for 3D object class recognition. Before the study, state-of-the-art volumetric representations for 3D data and already existing CNNs for this purpose were reviewed. A set of representations were

proposed to conduct this study, as well as a new architecture (inspired by the success of the existing and reviewed ones). The networks were trained using the ModelNet-10 dataset, whose models were adapted to our representations, and also augmented to simulate the aforementioned adverse conditions of real-world scenes, e.g., noise and occlusions.

As a result of the experimentation we can draw the following main conclusion: the volumetric representation itself has a huge impact on the performance of the network in terms of accuracy. On the one hand, the adaptive tensor exhibited not only better accuracy results than the fixed one, but it also introduced occlusion and noise robustness to some extent. On the other hand, the binary occupancy measure outperformed the normalized density one, fostering the less-is-better principle. In conclusion, this study provides a reasonable insight about the effect of 3D data representation in this kind of networks. In addition, it proves that taking into account real-world conditions is a matter of utmost importance when training these networks with synthetic datasets.

Furthermore, we compared our approaches with a state-of-the-art method: PointNet, which features a different approach for object recognition, using a deep network based on MLPs instead of convolutions. We found out that PointNet's approach achieves better base accuracy and noise resilience; however, it is outperformed by CNN-based approaches when dealing with occlusions. It is important to remark that occlusions are one of the main problems of real-world scenes.

Following on this work, we plan to improve the study by including more volumetric representations and improving the existing ones. For instance, applying orientation estimation methods to the adaptive grid in order to better fit the input cloud and find a consistent alignment throughout the models would probably yield an improvement. Another possible addition could be extending the occupancy computation to take into account the actual surface of the object, i.e., triangulating the point cloud and computing the amount of surface which intersects each voxel. Furthermore, this study has not taken into account the

efficiency. In this regard, it could be extended by analyzing the performance in terms of runtime of both representations and networks.

The source code used to generate the results claimed in this manuscript can be downloaded from the corresponding *GitHub* [2] repository.

Acknowledgments

This work has been supported by the Spanish Government DPI2013-40534-R grant for the SIRMAVED project, also supported with FEDER funds. This work has also been funded by the grant "Ayudas para Estudios de Máster e Iniciación a la Investigación" from the University of Alicante. Experiments were made possible by a generous donation of hardware from NVIDIA (Tesla K20 and K40).

References

[1] R. B. Rusu, S. Cousins, 3d is here: Point cloud library (pcl), in: Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE, 2011, pp. 1–4.

[2] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, J. Xiao, 3d shapenets: A deep representation for volumetric shape modeling, in: Proc. CVPR, to appear, Vol. 1, 2015, p. 3.

[3] S. Song, J. Xiao, Deep sliding shapes for amodal 3d object detection in RGB-D images, CoRR abs/1511.02300.
URL `http://arxiv.org/abs/1511.02300`

[4] D. Maturana, S. Scherer, 3d convolutional neural networks for landing zone detection from lidar, ICRA, 2015.

[2] `https://github.com/Blitzman/cviu-si-dl-study`

[5] R. Socher, B. Huval, B. Bath, C. D. Manning, A. Y. Ng, Convolutional-recursive deep learning for 3d object classification, in: Advances in Neural Information Processing Systems, 2012, pp. 665–673.

[6] L. A. Alexandre, 3d object recognition using convolutional neural networks with transfer learning between input channels, in: Proc. the 13th International Conference on Intelligent Autonomous Systems, 2014.

[7] S. J. Pan, Q. Yang, A survey on transfer learning, Knowledge and Data Engineering, IEEE Transactions on 22 (10) (2010) 1345–1359.

[8] D. C. Cireşan, U. Meier, J. Schmidhuber, Transfer learning for latin and chinese characters with deep neural networks, in: Neural Networks (IJCNN), The 2012 International Joint Conference on, IEEE, 2012, pp. 1–6.

[9] N. Höft, H. Schulz, S. Behnke, Fast semantic segmentation of rgb-d scenes with gpu-accelerated deep neural networks, in: KI 2014: Advances in Artificial Intelligence, Springer, 2014, pp. 80–85.

[10] H. Schulz, S. Behnke, Learning object-class segmentation with convolutional neural networks., in: ESANN, 2012.

[11] J. Wang, J. Lu, W. Chen, X. Wu, Convolutional neural network for 3d object recognition based on rgb-d dataset, in: Industrial Electronics and Applications (ICIEA), 2015 IEEE 10th Conference on, IEEE, 2015, pp. 34–39.

[12] M. Schwarz, H. Schulz, S. Behnke, Rgb-d object recognition and pose estimation based on pre-trained convolutional neural network features, in: Robotics and Automation (ICRA), 2015 IEEE International Conference on, IEEE, 2015, pp. 1329–1335.

[13] D. Maturana, S. Scherer, Voxnet: A 3d convolutional neural network for real-time object recognition.

[14] H. Su, S. Maji, E. Kalogerakis, E. G. Learned-Miller, Multi-view convolutional neural networks for 3d shape recognition, in: Proc. ICCV, 2015.

[15] B. Shi, S. Bai, Z. Zhou, X. Bai, Deeppano: Deep panoramic representation for 3-d shape recognition, Signal Processing Letters, IEEE 22 (12) (2015) 2339–2343.

[16] N. Sedaghat, M. Zolfaghari, T. Brox, Orientation-boosted voxel nets for 3d object recognition, arXiv preprint arXiv:1604.03351.

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

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958.

[19] K. Lai, L. Bo, X. Ren, D. Fox, A large-scale hierarchical multi-view rgb-d object dataset, in: Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE, 2011, pp. 1817–1824.

[20] A. Singh, J. Sha, K. S. Narayan, T. Achim, P. Abbeel, Bigbird: A large-scale 3d database of object instances, in: Robotics and Automation (ICRA), 2014 IEEE International Conference on, IEEE, 2014, pp. 509–516.

[21] B. Li, Y. Lu, C. Li, A. Godil, T. Schreck, M. Aono, M. Burtscher, H. Fu, T. Furuya, H. Johan, et al., Shrec'14 track: extended large scale sketch-based 3d shape retrieval, in: Eurographics Workshop on 3D Object Retrieval, 2014, pp. 121–130.

[22] S. Choi, Q. Zhou, S. Miller, V. Koltun, A large dataset of object scans, CoRR abs/1602.02481.

[23] M. Gschwandtner, R. Kwitt, A. Uhl, W. Pree, Blensor: blender sensor simulation toolbox, in: Advances in Visual Computing, Springer, 2011, pp. 199–208.

[24] C. R. Qi, H. Su, K. Mo, L. J. Guibas, Pointnet: Deep learning on point sets for 3d classification and segmentation, arXiv preprint arXiv:1612.00593.