A Network Architecture for Point Cloud Classification via Automatic Depth Images Generation

Riccardo Roveri[1†], Lukas Rahmann[1†], A. Cengiz Öztireli[2*], Markus Gross[1]
[1]Department of Computer Science, ETH Zürich
[2]Disney Research Zürich
[†]Equal contribution from both authors
{rroveri, grossm}@inf.ethz.ch, lrahmann@student.ethz.ch, cengiz.oztireli@disneyresearch.com

Abstract

We propose a novel neural network architecture for point cloud classification. Our key idea is to automatically transform the 3D unordered input data into a set of useful 2D depth images, and classify them by exploiting well performing image classification CNNs. We present new differentiable module designs to generate depth images from a point cloud. These modules can be combined with any network architecture for processing point clouds. We utilize them in combination with state-of-the-art classification networks, and get results competitive with the state of the art in point cloud classification. Furthermore, our architecture automatically produces informative images representing the input point cloud, which could be used for further applications such as point cloud visualization.

1. Introduction

Point clouds are an important and common representation of 3D data. In this paper, we tackle point cloud classification: being able to automatically classify point clouds is a challenging task that can have impact in many other problems in Computer Vision and Graphics, such as scene understanding and surface reconstruction.

Even though 3D scanners are becoming cheaper and more available, 2D images still represent the majority of our graphical information. Thanks to significantly large image datasets [4], and a growing interest in the research community, Convolutional Neural Networks (CNNs) for image classification have been well studied and achieved state of the art results. Inspired by their high quality results, we build a novel neural network architecture which allows us to exploit the strengths of 2D image based CNNs for classifying 3D point clouds. Unlike some very recent deep learning methods [17, 13, 22], which handle unordered 3D data by directly processing and classifying them, our idea is to design a set of trainable network components that automatically transform the 3D input to informative 2D images, which are then input to image classification networks. Contrary to previous works [23] that classify 3D meshes by exploiting rendered images, in our method, the images are not generated in a pre-processing step, but rather learned within the network.

In particular, our completely differentiable architecture first intrinsically predicts one or multiple views, which are informative about the shape and features of the input point cloud. Secondly, another differentiable module generates the corresponding depth images of the point cloud rendered from those views. These depth images are produced by extending the work presented in [21] to handle point clouds with multiple layers of depth, occlusions and overlapping structures. Finally, a third component combines the images and uses an image classification CNN [9] to classify them. Thanks to the generated depth images and high performance of CNNs for image classification, we obtain competitive classification results to the recent methods in the field. Furthermore, the views intrinsically generated by our network can be extracted as an additional output at testing time, and used for point cloud visualization.

To summarize, the contributions of this paper are the following:

- We propose a novel neural network architecture for point cloud classification that achieves results competitive with the state of the art, even for difficult noisy datasets. The key idea is to automatically transform the unordered 3D points to informative 2D images and exploit the well studied image based classification network architectures (and their pre-trained weights on large image datasets).
- Our architecture produces one or a set of informative depth images of the point cloud, by predicting meaningful view directions. We illustrate that the learned

[*] Work done while at ETH Zürich

view directions and the corresponding depth images can be used for other applications, such as point cloud visualization.

- We propose a fully differentiable module for generating depth images of point clouds representing full 3D objects with occluded points, by integrating a point cloud culling strategy. This module can be used in further tasks and architectures that work with point clouds.

2. Related Work

Deep Learning on Point Cloud Data Point clouds are a common instance of 3D data, and various deep learning architectures have been proposed in order to handle them. The most straightforward approach is to convert the point cloud to a uniform voxel grid and use CNN based methods for volumetric representations, such as the methods presented in [18, 27, 16, 2, 26]. While transforming the point cloud to a voxel grid allows to feed regularly structured data to the network, the main disadvantage of these techniques is that they are computationally expensive, limiting the resolution of the point cloud. Some attempts have been proposed in order to overcome the voxel grid resolution issue, for example by using an octree structure [20, 25], employing field probing filters [14], or exploiting the sparseness of the problem via voting schemes [24]. However, the low resolution nature of voxel grids still constraints the size of the processed point cloud.

Instead of converting the point cloud to a voxel grid, some new works have presented methods to directly process unordered point sets, achieving state of the art results in point cloud classification, comparable to methods which classify volumetric objects [16, 27, 18] and meshes [12]. The authors of PointNet [17] propose a network architecture to respect properties such as invariance to permutations and tranformations of the input points. In the recently published work [22], CNNs are generalized from grids to general graphs using edge-dependant filters. Similarly, in Kd-Networks [13], a concurrent work to ours, kd-trees are used as underlying graphs to simulate CNNs. Finally, the concurrent work PointNet++ [19] improves the original PointNet by applying the network recursively on a nested partitioning of the input point set. Like these last set of approaches, our method takes an unordered point cloud as input, which can also be of high resolution. Contrary to them, instead of tackling classification directly on the point cloud, we first extract a set of 2D depth images, and then exploit well studied CNN based image classification methods to classify point clouds.

Exploiting Multiple Views on 3D Data Many deep learning methods utilize multiple 2D views of 3D data in order to learn more complex features. For example, in [6] and [5] the authors show how adding additional views of the human body produces better results in estimating their shape. In MVCNN [23], a 3D shape model is rendered with different virtual cameras from fixed view points, and the resulting images are combined with a view pooling operation and classified with a CNN based architecture. In the recent [11], views of 3D meshes are rendered from selected viewpoints in an initial step, and fed to a network architecture which segments the meshes using projective CNNs to project images onto the shape surface representation. These approaches require a preprocessing step where the input meshes are rendered from a set of views, using standard mesh rendering pipelines. Contrary to these works, we introduce a differentiable module for rendering point cloud data from different views on-the-fly from the input, which allows the network to automatically learn the most useful view directions.

Image recognition using CNNs Our method is related to image based CNN architectures, as we classify point clouds by first automatically extracting 2D images. CNNs have produced state of the art results in image recognition and related tasks e.g. [3, 7, 8, 9]. In particular, large image datasets available [4] allow CNNs to learn features that are general and suitable for different operations. For 3D data, such large datasets are not available and harder to obtain, which lies behind our idea of extracting 2D features from 3D data. In this work, we classify our extracted 2D views with ResNet [9], and utilize ImageNet [4] as a dataset for pre-training.

Rendering Depth Images In Neural Networks Rendering 2D images from 3D geometry within a neural network is an interesting task that could have impact in many computer vision applications. Spatial Transformer Networks [10] presents a differentiable module for applying transformations to a feature map. By applying a 3D affine matrix and flattening the result, their method can produce a 2D projection of the 3D voxel grid input. In [18], the authors also propose a differentiable module based on anisotropic kernels to generate 2D images using voxel grids as input. In OpenDR [15], a differentiable renderer for triangle-based geometry is presented. While the simple 3D to 2D projection of Spatial transformer Networks [10] does not deal with rendering, [18] requires a conversion from point clouds to a low resolution representation and OpenDR [15] works on triangles, in our work we aim at generating depth images from unordered point clouds. In addition, our work focuses on intrinsically learning a projection direction. A key component of our method is a differentiable module to generate depth images from the points, which can be inserted into any neural network architecture. We took [21] as a basis,

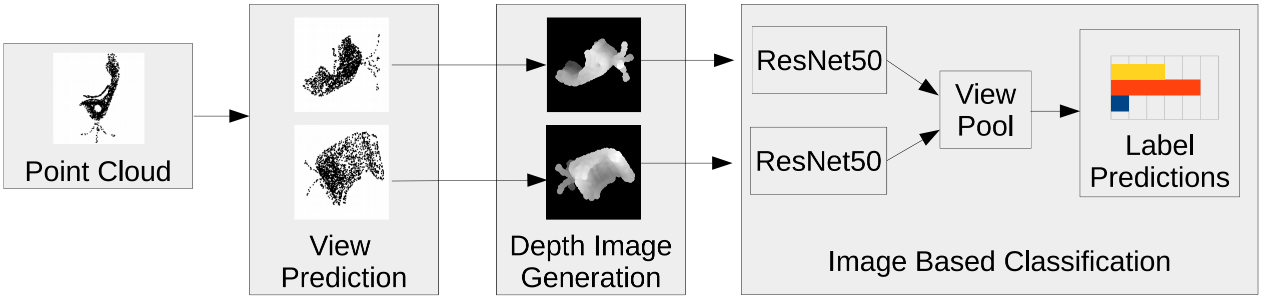

Figure 1. An overview of our network architecture. Given an input point cloud of a chair, two informative views are predicted, the corresponding depth images are generated and fed to the image classification module.

where the authors create depth field images by projecting the points onto an image plane. However, their method is designed to handle local patches of points, where the geometry is well represented with a height field without occluded parts. We extend it for generating informative depth images of full point clouds with multiple layers of depth, by carefully interpolating point depth values.

3. Network Architecture

3.1. Overview

Instead of directly classifying a point cloud, we designed a network architecture which automatically transforms the 3D input into a set of informative 2D depth images, and then solves the problem of classifying them. The view directions for generating the depth images are learned in an unsupervised manner, thus predicted with the goal of maximizing the classification accuracy. The main advantage of this approach, compared to directly processing the 3D points as in [17], is that it allows us to exploit the well studied deep learning architectures for image classification, which have been proven to achieve state-of-the-art results. Moreover, in addition to outputting a class label prediction for the input point cloud, our network intrinsically learns to predict one or a set of informative view directions and generate the corresponding 2D depth images, which could be used for other applications e.g. for 3D object recognition or point cloud visualization.

Given an input point cloud P and a desired number of views K, our pipeline is to first choose K views directions, generate the correspondent depth images and then utilize them to classify the point cloud. Our network architecture is thus composed of three modules: the first one takes the input point cloud coordinates as input and predicts K direction vectors (Section 3.2); the second module receives the input point cloud coordinates and the K directions and generates the depth images accordingly (Section 3.3); finally, the third module combines the K depth images and produces a vector representing the prediction labels for classes (Section 3.4).

We train the three modules jointly within a single architecture, using a softmax cross entropy loss on the class labels, provided as ground truth. Notice that we do not include a loss on the view directions nor on the generated depth images. Figure 1 shows an overview of the network architecture, for the case where $K = 2$.

3.2. View Prediction

The first module of our architecture receives the point cloud P as input and produces K view directions, where K is a parameter chosen by the user. In particular, we want to estimate the K camera-pose matrices which represent 3D rotations to transform the point cloud in order to perform an orthogonal projection. Inspired by [21], we start from the recent method PointNet [17], which proposes a network architecture that allows for processing unordered 3D point sets, like our input. The prediction of the views should respect crucial properties such as invariance to permutations of the input data and invariance under transformations. PointNet achieves input permutations invariance through a max pooling layer that approximates a symmetric function, and transformation invariance by predicting an affine matrix applied to the input. Finally, a fully connected layer creates a global descriptor used for classification.

In our work, we utilize a separate PointNet architecture for each of our K views, modifying the final fully connected layer to produce a 6D vector, representing the camera view vector v_k and the up-axis u_k of the kth camera-pose matrix. We build C_k, the camera-pose matrix for the kth view, by setting $w_k = v_k \times u_k$ and $C_k = \left[w_i^T; u_i^T; v_i^T\right]$.

We finally multiply the input point cloud sequentially with every camera-pose matrix, producing K rotated point clouds. After the transformation, the p_x and p_y coordinates of a point $p \in P$ represent its image coordinates, and the p_z coordinate is its depth.

Note that, due to memory limitations, we utilize a subsampled version of the input point cloud (keeping 12.5% of the original points) to estimate the camera-pose matrices. See Section 4.1 for more details on the implementation. Figure 2 shows a diagram of the view prediction module.

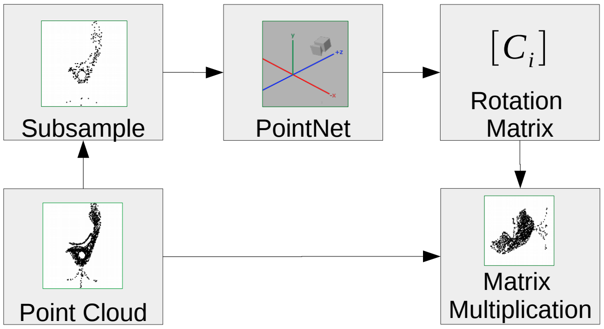

Figure 2. Our view prediction module. The camera-pose matrix parameters are estimated from a downsampled version of the input, and the original points are rotated accordingly.

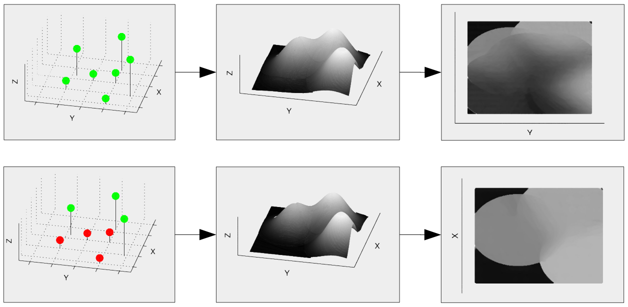

Figure 3. (Top) The Gaussian interpolation of [21], where all the points are interpolated together. (Bottom) Our Gaussian interpolation, where the red points are filtered out because their depth is too different than the max_D values of the pixels. The green points are interpolated.

3.3. Depth Image Generation

In this network component, the goal is to generate a depth image for an input point cloud using differentiable operations. In [21], the authors present a differentiable layer to create a distance field image from a point cloud by interpolating the depth values of the points on the image plane using Gaussian interpolation. Since they project every point of the point cloud to the image plane and interpolate their distances, their approach is mostly suitable for generating depth images of small patches of points lying on a single sheet without overlapping structures, but does not produce useful depth images when the point cloud contains structures on different depth layers. In that case, indeed, points with a large difference in the depth coordinate may be projected to closeby pixels on the image, and their depth values will be averaged together, leading to skewed geometry representations as shown in Figure 3 (top).

In this paper, we propose an extension that is suitable for point clouds representing full objects, with points lying also on occluded multiple layers. We thus aim at producing depth images which properly approximate a rendering of a surface passing through the points, which can be more reliably classified by our final image classification component. In particular, a depth image should present clear edges where the depth changes abruptly, and continuous values in smooth parts of the object.

The main idea of our depth image generation method is to apply a bilateral-filering-like interpolation to obtain point cloud culling. In practice, instead of considering all the points in the point cloud, we segment the points belonging to the farthest surface layers from the image plane, and interpolate only their depths. We get the final image f by first computing a maximum depth value max_D for every pixel c, representing the maximum depth of the points $p \in P$ which are close enough to c when projected on the image plane. We define the subset $P'(c)$ as all the points $p \in P$ close enough to c on the image plane, given a threshold δ_1 :

$$P'(c) = \{p \in P | \ \|(c_x, c_y) - (p_x, p_y)\| < \delta_1\}. \quad (1)$$

It follows that:

$$max_D(c) = \max\{p_z| \ p \in P'(c)\}. \quad (2)$$

If $P'(c)$ is empty, no points will be projected closeby c, so we set the final value $f(c)$ for the pixel to zero. This step implicitly introduces a cutoff distance of δ_1 to the final Gaussian interpolation of our image generation procedure, allowing us to produce depth images with clear hard edges at the border of the objects. In case $P'(c)$ is not empty, in order to compute the final value $f(c)$ for a pixel, we consider only the points which have a depth value close enough to $max_D(c)$, and interpolate their depths (Figure 3, bottom). This ensures that our generated image has hard edges where the depth changes abruptly and smoother variations elsewhere. For a chosen threshold δ_2, we define a new subset of points $P''(c)$:

$$P''(c) = \{p \in P| \ |max_D(c) - p_z| < \delta_2\}. \quad (3)$$

For a pixel c, we apply Gaussian interpolation on the depth values of the points in $P''(c)$ as follows:

$$f(c) = \frac{1}{W} \sum_{p \in P''(c)} g((c_x, c_y), (p_x, p_y))p_z, \quad (4)$$

with a normalization term W:

$$W = \sum_{p \in P''(c)} g((c_x, c_y), (p_x, p_y)), \quad (5)$$

and a Gaussian function g:

$$g((x, y), (x', y')) = e^{\frac{-(x-x')^2-(y-y')^2}{2\sigma^2}}, \quad (6)$$

where σ influences the smoothness of the generated depth images, as can be seen in Figure 4.

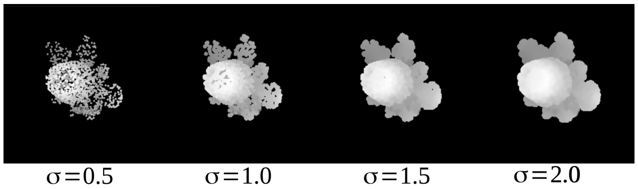

Figure 4. Depth images of a flower pot generated with different values for σ.

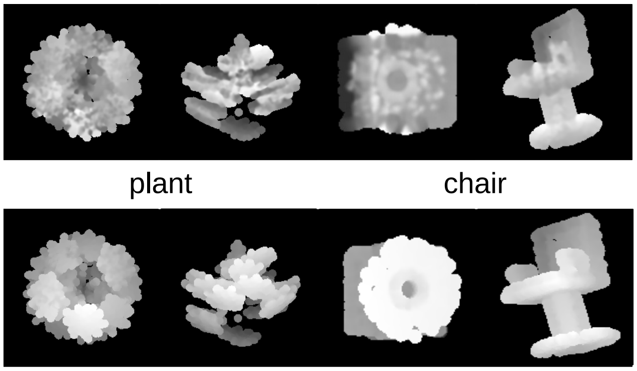

Figure 5. Depth images of a flower pot and a chair from two different views each, generated using the simple projection and Gaussian interpolation from [21] (above) and using our depth image generator (below).

We generate a depth image for each of the K point clouds passed from the previous module. Notice that since we define max_D with the maximum depth, the resulting views will be rendered for a camera placed at $[0, 0, 1]$ and pointing at $[0, 0, 0]$, assuming the point cloud lies in $[-1, 1]^3$. This module was implemented as a custom layer in TensorFlow and CUDA, see Section 4.1 for implementation details. The gradients of this custom module are shown in the supplementary material.

Figure 5 shows various depth images produced with our module and with the simple projection and Gaussian interpolation from [21]. Our results better represent the depth of the object, especially featuring sharper edges at discontinuities in the depth field.

3.4. Image Based Classification

Our final module takes the K depth images generated by the previous layer and implements a classification network for images. In particular, we utilize K ResNet50 [9] architectures that share variables. Similar to MVCNN [23], which deals with classifying images from different fixed views for 3D objects, we include a max pooling operation before regressing to a denser layer of classification logits. This allows the network to share variables between the ResNet50 architectures and thus learn features which require multiple images. We place the dense classification layer after the max pooling operation. For the ResNet50 architectures, we make use of pre-trained weights as initialization, trained on ImageNet [4] dataset.

	3D overall	y-Axis overall	3D class	y-Axis class
Ours, 1 View	0.854	0.873	0.815	0.828
Ours, 2 Views	0.869	0.884	0.829	0.851
Ours, 4 Views	**0.872**	0.885	**0.830**	0.856
PointNet	0.855	**0.892**	0.805	**0.862**

Table 1. Classification results of our architecture with 1, 2 and 4 views, and the PointNet [17] method, on a dataset augmented with random rotations (3D) and augmented only with random rotations around the vertical axis (y-Axis). Both the instance-based accuracy (overall) and the class average (class) are shown.

4. Results

4.1. Implementation, Parameters and Timing

In the view prediction component, we subsample original point clouds consisting of 2048 points to 256 points. All the batch normalization layers in PointNet were trained using a batch normalization decay of 0.9. In the depth image generation component, we set $\sigma = 2.0$, $\delta_1 = 1.4\sigma$, and $\delta_2 = I/12$, where I is the image size and is 229. The depth image generation layer was implemented as two native ops (gradient and forward pass) in TensorFlow using the CUDA backend. For the image based classification component, we used the preset values from the TensorFlow Slim library.

In order to train the network, we used the Adam optimizer with an initial learning rate of 0.0001 lowered every 50000 steps by a percentage of 5% (if the number of views $K = 1$), 10% (if $K = 2$), and 20% (if $K = 4$). The batch size varied between the models (128 for one view, 64 for two views and 32 for four views), due to memory reasons.

In our implementation tested on a GeForce GTX 1080 Ti graphics card, the forward-pass and backward-pass through the depth image generation layer take around 2 seconds (1.323 seconds for the forward-pass, 0.752 seconds for the backward-pass), using a batch size of 512, and projecting 2048 points to images of 229x229 pixels. In the complete graph, this corresponds to 10% of the computation time for the forward pass and 7% for the backward pass. The resulting training time of our single view architecture on randomly rotated point clouds is comparable to PointNet (about 8 hours). The training time increases with a sublinear dependency on the number of views, as the convergence is faster with multiple views.

4.2. Point Cloud Classification

We evaluate both PointNet and our network variations on the ModelNet40 [27] benchmark for shape classification, composed of CAD models labelled in 40 classes and separated between training (9843 models) and testing (2468).

For our method, we generated the point clouds from the shape dataset by uniformly sampling 2048 points on each model. For PointNet [17], we used the most recent version of the original implementation by the authors with the default settings (1024 points), as using 2048 points did not improve the results. Both for our method and for PointNet, we augment the training and testing models in the dataset by applying two different strategies: rotating the point clouds randomly in every direction, and rotating the point clouds randomly only around the vertical axis. Similarly to PointNet, in our training data we augment the point clouds by adding random Gaussian of noise of $\sigma = 0.001$. We let our method train until convergence, and present the average of the testing results from 5 evaluations.

In Table 1, we show the results of our classification using 1, 2 and 4 views and compare them to the state of the art PointNet [17] point cloud classifier, for both the dataset augmented with random rotations in every direction (3D) and the dataset augmented with random rotations only around the vertical axis (y-Axis). Both the instance-based accuracy (overall) and the class average accuracy (class) are presented. The best value for each column is highlighted.

The first three rows show how our architecture profits from estimating multiple depth images, thanks to our image based classification network that combines features from multiple views to optimize the outcome. Our best results are obtained using 4 views, for both datasets.

While our 4 views architecture obtains slightly worse results than PointNet in the dataset with objects aligned along the vertical axis, our results vary less between the two datasets. In particular, in the more difficult dataset with randomly rotated objects, we outperform PointNet with our 2 and 4 views architectures, and obtain comparable results with a single view. Our intuition is that our generated depth images are more informative and result in less source of confusion while learning. Image classification is a better studied problem than 3D point cloud classification, thus utilizing state of the art image classification networks (and their pre-trained weights on large datasets) on properly estimated depth images allows us to obtain better results, starting from the same raw input.

In our experiments, adding more views than 4 did not help improving the results. We believe that this limit is due to the sparsity of the point clouds, which do not contain very detailed features. Thus, our 4 views can already include the majority of the structure of the object representations in the dataset.

Note that all the recently proposed or concurrent techniques for classifying 3D points [17, 19, 18, 22, 13] present results obtained on the easier dataset, where the objects are aligned with the y-axis. The concurrent work PointNet++ [19] obtains better classification results than the original PointNet by about 2%, while the volumetric variant of the recent work [18] obtains results comparable to PointNet. As future work, it would be interesting to test if they generalize to the dataset with random rotations. Finally, there exist works which use 3D meshes instead of 3D point clouds and obtain better classification results. In [23], the best results are obtained by rendering 80 views of 3D shape models; their views, though, are fixed and not learned, and their input 3D meshes are more detailed than our point clouds. Generating meshes from our sparse and noisy point clouds with triangulation or surface reconstruction methods would lead to meshes of significantly lower quality, leading to inferior results for mesh rendering based methods in this case.

	Learned	PCA
1 View	0.854	0.844
2/3 Views (Learned/PCA)	0.869	0.850

Table 2. Instance-based classification results of our simpler PCA alternative with 1 and 3 views (PCA), compared to our original architecture results for 1 and 2 views on the dataset augmented with random rotations (Learned).

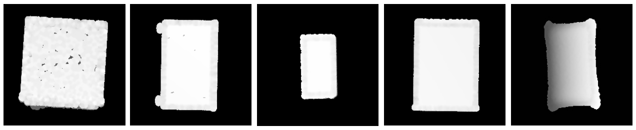

Figure 6. Examples of images generated by our network with PCA.

4.3. Comparisons to Simpler Alternatives

In order to quantitativaly evaluate the impact of our automatic view estimation, we trained our architecture by substituting the view prediction module with a selection based on the PCA axes. We performed this experiment with a single view (projecting onto the plane spanned by the directions of the two largest PCA components), and 3 views (projecting on each plane spanned by the PCA directions). Notice that the obtained PCA views, thus the final accuracy, are equivalent in both the dataset augmented with random rotations and the dataset augmented with random rotations only around the y-Axis. In Table 2 we compare the obtained instance-based accuracy results (PCA) to the ones of our 1 and 2 views original architectures on the more difficult dataset with random rotations (Learned). One can notice how, already in the more difficult dataset, the PCA-based average accuracies are lower than our original 1 and 2 views results. In Figure 6, we show examples of PCA views of objects from different classes produced by the network, demonstrating how they are often ambiguous. Important features of the objects can indeed be hidden by their large surfaces, and, in case of isotropic point clouds, the PCA views are equivalent to random views (see our Supplementary Material for more examples).

Furthermore, we trained our architecture by substituting

# Train.	87	104	173	163	149	64	124	197	240	88	231	138	167	680	124	475	128	200	149	392	200	200	90	103	106
± Acc.	-0.03	-0.02	-0.02	-0.02	-0.01	-0.01	-0.01	-0.01	-0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03	0.03	0.04	0.05
	wardrobe	radio	bench	tent	laptop	bowl	stairs	car	plant	person	piano	curtain	cone	sofa	lamp	vase	sink	night stand	flower pot	table	desk	dresser	stool	xbox	bathtub

Table 4. Difference of obtained accuracy between our original method and the random views alternative (± Acc.) and number of training examples (# Train.), per class. Positive: the proposed method is better.

	3D Learned	y-Axis Learned	Random
1 View	0.854	0.873	0.849
2 Views	0.869	0.884	0.859
4 Views	0.872	0.885	0.868

Table 3. Instance-based classification results of our simpler random views alternative (Random), compared to our original architecture results on the dataset augmented with random rotations (3D, Learned) and the dataset augmented with random rotations only around the y-Axis (y-Axis, Learned).

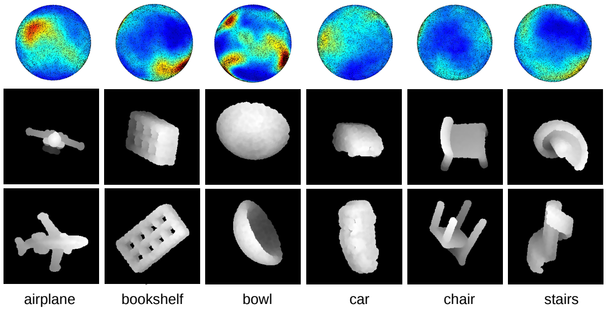

Figure 7. Learned view density functions (top), depth images generated by our network corresponding to the least likely learned view (center), and the most likely learned view (bottom), for six testing point clouds.

the view prediction module with a random view selection, for 1, 2 and 4 views. In Table 3, we compare the obtained instance-based results (Random) with the ones of our original architectures in both datasets (3D and y-Axis, Learned). Notice that the results of the random views alternative are equivalent in both datasets, as the views are randomly sampled in the 3D space. The results obtained by our original architectures are better than the ones of random views for any number of views, especially for the dataset augmented with rotations around the y-Axis. For the more difficult dataset augmented with random rotations, the improvement given by our learned views is smaller. We believe that the gap could be larger if the ModelNet40 dataset would not present some commonly known limitations [23, 1] such as ambiguities between pairs of classes (especially for low resolution inputs like ours) and classes with little training examples. In Table 4, we present the difference of obtained accuracy between our original method and the random views alternative (± Acc.), averaged between the 1, 2 and 4 views results, for the classes where the absolute difference was at least 1%. For each class, we additionally show the number of training examples (# Train.). The classes where the random views alternative performed better (i.e., where the values are smaller than zero) are often the ambiguous ones such as plant (confused with flower pot and vase), wardrobe (confused with bookshelf) and radio (confused with regular objects such as glass box), and have in general small number of training examples per class (on average 145, while the classes where our original method works better have on average 273 examples). We thus expect our method, based on learned views, to work best with a large number of training examples per class. We refer to the Supplementary Material for the individual accuracy results for 1, 2 and 4 views used for Table 4.

4.4. View Selection and Visualization

Our network outputs a set of views and corresponding depth images as additional information. The depth images generated represent informative 2D views of the 3D input, and can be utilized for applications such as visualizing a point cloud. In order to visually evaluate the quality of our learned views, we feed a set of test point clouds to our single view architecture multiple times, always with a different random rotation, and plot the distribution of the learned views for each point cloud. Each point cloud is fed 10000 times to the network, and the resulting learned views, with respect to the original orientation of the point cloud, are sampled on a sphere.

In Figure 7 (top) the density of these learned views for six testing models are presented (red: high, blue: low). The density functions contain peaks and further regions with very low values. Hence, our views are optimized for different objects, and not random or constant. For most objects, multiple views can be considered appropriate for classification. Thus, we do not expect our density functions to present only a single sharp peak, but rather smoother regions of high values.

For each test point cloud, we sample the view corresponding to the highest value of the views density (i.e. the most likely view estimated by our network), and show the depth image generated by our network for that view in Figure 7 (bottom). Similarly, Figure 7 (center) presents the

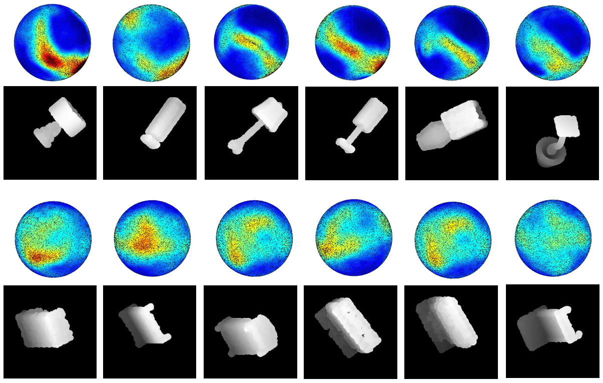

Figure 8. Learned view density functions and the depth images corresponding to the view with highest probability for two classes of objects (lamps and beds).

depth images corresponding to the view with the lowest probability. Our view estimation procedure outputs depth images that clearly expose distinguishing features, making the classes easily recognizable. For example, the legs of the chair, the tail of the airplane, the internal structure of the bookshelf, and the borders of the bowl are visible from the learned views, while hidden for a view with low probability of selection. Similarly, the shapes of the stairs and the car are fully visible for the view with high probability of selection by our architecture, while only partially for a low probability view.

In Figure 8, further view density functions and depth images for the highest probability views are presented, for two classes of objects (lamps and beds). It is interesting to see how the learned views are similar for objects of the same class, but different for objects of different classes. This shows that the network specializes the views according to the classes.

Figure 9 shows examples of depth images generated by our network with two views, for different classes of objects. To generate these results, the test point clouds were fed to the network 10 times with random rotations, and the run which produced the highest single softmax prediction was considered. In most cases, our network predicts two views which complement each other, providing an even clearer overview of the point cloud compared to the views given by our single view architecture. This is a consequence of the view pooling operation in our network, which combines features from different views. For example, the two depth images of the bed, piano, and bookshelf are a side view and a top/down view, the images of the chair show its front and back, and those of the cone contain the bottom hole and the pick at the top.

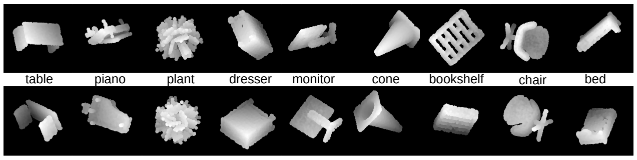

Figure 9. Examples of generated depth images from the two views architecture. The two images (top and bottom) complement each other, providing a more informative overview of the object, as compared to a single view.

5. Conclusion

We propose a novel neural network architecture for point cloud classification, which obtains results competitive with the state of the art for raw point clouds as the input. Our key idea is to automatically transform the input point cloud to one or more depth images, which can be combined and classified by a CNN classification module. The high performance of image based CNNs, and the large availability of data to train them, makes classification on our images better than considering only the 3D data.

In the future, we would like to explore further applications of this view learning and depth images generation approach. First of all, more experiments on point cloud visualization could be performed. Properly visualizing a point cloud is not a trivial task due to occlusions, lack of detailed features and sparse data, and we believe that our depth image generation method can be a useful representation. Another possible interesting extension would be to use our depth image generation layer as an autoencoder or for unsupervised learning. Finally, it can be adapted to generative adversarial networks (GANs) e.g. for point cloud segmentation.

Acknowledgments Riccardo Roveri is supported by the Swiss National Science Foundation (grant No. 200021_146227/2).

References

[1] V. Arvind, A. Costa, M. Badgeley, S. Cho, and E. Oermann. Wide and deep volumetric residual networks for volumetric image classification. *CoRR*, abs/1710.01217, 2017. 7

[2] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Generative and discriminative voxel modeling with convolutional neural networks. *CoRR*, abs/1608.04236, 2016. 2

[3] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014. 2

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 1, 2, 5

[5] E. Dibra, H. Jain, A. C. Öztireli, R. Ziegler, and M. H. Gross. Hs-nets: Estimating human body shape from silhouettes with

convolutional neural networks. In *Fourth International Conference on 3D Vision, 3DV 2016, Stanford, CA, USA, October 25-28, 2016*, pages 108–117, 2016. 2

[6] E. Dibra, H. Jain, A. C. Öztireli, R. Ziegler, and M. H. Gross. Human shape from silhouettes using generative hks descriptors and cross-modal neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, July 21-26, 2017*, 2017. 2

[7] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference in Machine Learning (ICML)*, 2014. 2

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 580–587, Washington, DC, USA, 2014. IEEE Computer Society. 2

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 1, 2, 5

[10] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu. Spatial transformer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc., 2015. 2

[11] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri. 3D shape segmentation with projective convolutional networks. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[12] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '03, pages 156–164, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. 2

[13] R. Klokov and V. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1, 2, 6

[14] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas. FPNN: field probing neural networks for 3d data. *CoRR*, abs/1605.06240, 2016. 2

[15] M. M. Loper and M. J. Black. OpenDR: An approximate differentiable renderer. In *Computer Vision – ECCV 2014*, volume 8695 of *Lecture Notes in Computer Science*, pages 154–169. Springer International Publishing, Sept. 2014. 2

[16] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, September 2015. 2

[17] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016. 1, 2, 3, 5, 6

[18] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2016. 2, 6

[19] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 2, 6

[20] G. Riegler, O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2

[21] R. Roveri, A. C. Öztireli, I. Pandele, and M. Gross. PointProNets: Consolidation of Point Clouds with Convolutional Neural Networks. *Computer Graphics Forum*, 2018. 1, 2, 3, 4, 5

[22] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1, 2, 6

[23] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015. 1, 2, 5, 6, 7

[24] D. Z. Wang and I. Posner. Voting for voting in online point cloud object detection. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015. 2

[25] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Trans. Graph.*, 36(4):72:1–72:11, July 2017. 2

[26] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016. 2

[27] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920. IEEE Computer Society, 2015. 2, 5