Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models

Roman Klokov
Skolkovo Institute of Science and Technology
roman.klokov@skoltech.ru

Victor Lempitsky
Skolkovo Insitute of Science and Technology
lempitsky@skoltech.ru

Abstract

We present a new deep learning architecture (called Kd-network) that is designed for 3D model recognition tasks and works with unstructured point clouds. The new architecture performs multiplicative transformations and shares parameters of these transformations according to the subdivisions of the point clouds imposed onto them by kd-trees. Unlike the currently dominant convolutional architectures that usually require rasterization on uniform two-dimensional or three-dimensional grids, Kd-networks do not rely on such grids in any way and therefore avoid poor scaling behavior. In a series of experiments with popular shape recognition benchmarks, Kd-networks demonstrate competitive performance in a number of shape recognition tasks such as shape classification, shape retrieval and shape part segmentation.

1. Introduction

As the 3D world around us is getting scanned and digitized and as the archives of human-designed models are growing in size, recognition and analysis of 3D geometric models are gaining importance. Meanwhile, deep convolutional networks (ConvNets) [15] have excelled at solving analogous recognition tasks for 2D image datasets. It is therefore natural that a lot of research currently aims at the adaptation of deep ConvNets to 3D models [36, 18, 4, 35, 34, 21, 31, 2, 3].

Such adaptation is non-trivial. Indeed, the most straightforward way to make ConvNets applicable to 3D data, is to rasterize 3D models onto uniform voxel grids. Such approach however leads to excessively large memory footprints and slow processing times. Consequently, works that follow this path [36, 18, 4, 35, 34, 16] use small spatial resolutions (e.g. $64 \times 64 \times 64$), which clearly lag behind grid resolutions typical for processing 2D data, and is likely to be insufficient for the recognition tasks that require attention to fine details in the models.

To solve this problem, we take inspiration from the long history of research in computer graphics and computational geometry communities [25, 10], where a large number of indexing structures that are far more scalable than uniform grids have been proposed, including kd-trees [1], octrees [19], binary spatial partition trees [28], R-trees [11], constructive solid geometry [22], etc. Our work was motivated by the question, whether at least *some of these indexing structures are amenable for forming the base for deep architectures*, in the same way as uniform grids form the base for the computations, data alignment and parameter sharing inside convolutional networks.

In this work, we pick one of the most common 3D indexing structures (a kd-tree [1]) and design a deep architecture (a *Kd-network*) that in many respects mimics ConvNets but uses kd-tree structure to form the computational graph, to share learnable parameters, and to compute a sequence of hierarchical representations in a feed-forward bottom-up fashion. In a series of experiments, we show that Kd-networks come close (or even exceed) ConvNets in terms of accuracy for recognition operations such as classification, retrieval and part segmentation. At the same time, Kd-networks come with smaller memory footprints and more efficient computations at train and at test time thanks to the improved ability of kd-trees to index and structure 3D data as compared to uniform voxel grids.

Below, we first review the related work on convolutional networks for 3D models in Section 2. We then discuss the Kd-network architecture in Section 3. An extensive evaluation on toy data (a variation of MNIST) and standard benchmarks (ModelNet10, ModelNet40, SHREC'16, ShapeNet part datasets) is presented in Section 4. We summarize the work in Section 5.

2. Related Work

Several groups investigated application of ConvNets to the rasterizations of 3D models on uniform 3D grids [36, 18]. The improvements include combinations of generative and very deep discriminative architectures [4, 35]. De-

spite considerable success in coarse-level classification, the reliance on uniform 3D grids for data representation makes scaling of such approaches to fine-grained tasks and high spatial representations problematic. To improve the scalability [34, 16] have considered sparse ways to define convolutions, while still using uniform 3D grids for representations.

Another approach [31, 21] is to avoid the use of 3D grids, and instead apply two-dimensional ConvNets to 2D projections of 3D objects, while pooling representations corresponding to different views. Despite gains in efficiency, such approach may not be optimal for hard 3D shape recognition tasks due to the loss of information associated with the projection operation. A group of approaches (such as spectral ConvNets [6, 2] and anisotropic ConvNets [3]) generalize ConvNets to non-Euclidean geometries, such as mesh surfaces. These have shown very good performance for local correspondence/matching tasks, though their performance on standard shape recognition and retrieval benchmarks has not been reported. Kd-networks as well as the PointNet architecture [20] work directly with points and therefore can take the representations computed with intrinsic ConvNets as inputs. Such configuration is likely to combine at least some of the advantages of extrinsic and intrinsic ConvNets, but its investigation is left for future work.

Aside from their connections to convolutional networks that we discuss in detail below, Kd-networks are related to recursive neural networks [30]. Both recursive neural networks and Kd-networks have tree-structured computational graphs. However, the former share parameters across all nodes in the computational tree graph, while sharing of parameters in Kd-networks is more structured, which allows them to achieve competitive performance.

Finally, two approaches developed in parallel to ours share important similarities. OctNets [23] are modified ConvNets that operate on non-uniform grids (shallow OctTrees) and thus share the same idea of utilizing non-uniform spatial structures within deep architectures. Even more related are graph-based ConvNets with edge-dependant filters [29]. Kd-networks can be regarded as a particular instance of their architecture with a kd-tree being an underlying graph (whereas [29] evaluated nearest neighbor graphs for point cloud classification). Kd-networks outperform both [23] and the setup in [29] on the ModelNet benchmarks suggesting that deep architectures based on kd-trees may be particularly well suited for coarse-level shape categorization.

3. Shape Recognition with Kd-Networks

We now introduce Kd-networks, starting with the discussion of their input format (kd-trees of certain size), then discussing the bottom-up computation of representations performed by Kd-networks, and finally discussing supervised parameter learning.

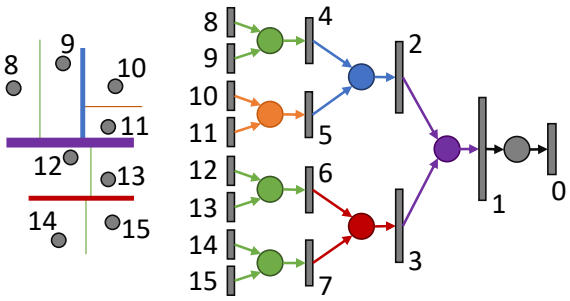

Figure 1. A kd-tree built on the point cloud of eight points (left), and the associated Kd-network built for classification (right). We number nodes in the kd-tree from the root to leaves. The arrows indicate information flow during forward pass (inference). The leftmost bars correspond to leaf (point) representations. The rightmost bar corresponds to inferred class posteriors $\mathbf{v}_0$. Circles correspond to affine transformations with learnable parameters. Colors of the circles indicate parameter sharing, as splits of the same type (same orientation, same tree level – three "green" splits in this example) share the transformation parameters.

3.1. Input

The new deep architecture (the Kd-network) works with kd-trees constructed for 3D point clouds. Kd-networks can also consider and utilize properties of individual input points (such as color, reflectivity, normal direction) if they are known. At train time, Kd-network works with point clouds of a fixed size $N = 2^D$ (point clouds of different sizes can be reduced to this size using sub- or oversampling). A kd-tree is constructed recursively in a top-down fashion by picking the coordinate axis with the largest range (span) of point coordinates, and splitting the set of points into two equally-sized subsets, subsequently recursing to each of them. As a result, a balanced kd-tree $\mathcal{T}$ of depth D is produced that contains $N-1 = 2^D-1$ non-leaf nodes.

Each non-leaf node $V_i \in \mathcal{T}$ is thus associated with one of three splitting directions d_i (along x, y or z-axis, i.e. $d_i \in \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$) and a certain split position (threshold) τ_i. A tree node is also characterized by the level $l_i \in \{1, .., D-1\}$, with l_i=1 for the root node, and l_i=D for tree leaves that contain individual 3D points. We assume that the nodes in the balanced tree are numbered in the standard top-down fashion, with the root being the first node, and with the ith node having children with numbers $c_1(i) = 2i$ and $c_2(i) = 2i+1$.

3.2. Processing data with Kd-networks

Given an input kd-tree $\mathcal{T}$, a pretrained Kd-network computes vectorial representations $\mathbf{v}_i$ associated with each node of the tree. For the leaf nodes these representations are given as k-dimensional vectors describing the individ-

ual points, associated with those leaves. The representations corresponding to non-leaf nodes are computed in the bottom-up fashion (Figure 1). Consider a non-leaf node i at the level $l(i)$ with children $c_1(i)$ and $c_2(i)$ at the level $l(i)+1$, for which the representations $\mathbf{v}_{c_1(i)}$ and $\mathbf{v}_{c_2(i)}$ have already been computed. Then, the vector representation $\mathbf{v}_i$ is computed as follows:

$$\mathbf{v}_i = \begin{cases} \phi(W_{\mathtt{x}}^{l_i}[\mathbf{v}_{c_1(i)};\mathbf{v}_{c_2(i)}] + \mathbf{b}_{\mathtt{x}}^{l_i}), \text{ if } d_i = \mathtt{x}\,, \\ \phi(W_{\mathtt{y}}^{l_i}[\mathbf{v}_{c_1(i)};\mathbf{v}_{c_2(i)}] + \mathbf{b}_{\mathtt{y}}^{l_i}), \text{ if } d_i = \mathtt{y}\,, \\ \phi(W_{\mathtt{z}}^{l_i}[\mathbf{v}_{c_1(i)};\mathbf{v}_{c_2(i)}] + \mathbf{b}_{\mathtt{z}}^{l_i}), \text{ if } d_i = \mathtt{z}\,, \end{cases} \tag{1}$$

or in short form:

$$\mathbf{v}_i = \phi(W_{d_i}^{l_i}[\mathbf{v}_{c_1(i)};\mathbf{v}_{c_2(i)}] + \mathbf{b}_{d_i}^{l_i})\,. \tag{2}$$

Here, $\phi(\cdot)$ is some non-linearity (e.g. REctified Linear Unit $\phi(a) = \max(a, 0)$), and square brackets denote concatenation. The affine transformation in (1) is defined by the learnable parameters $\{W_{\mathtt{x}}^{l_i}, W_{\mathtt{y}}^{l_i}, W_{\mathtt{z}}^{l_i}, \mathbf{b}_{\mathtt{x}}^{l_i}, \mathbf{b}_{\mathtt{y}}^{l_i}, \mathbf{b}_{\mathtt{z}}^{l_i}\}$ of the layer l_i. Thus, depending on the splitting direction d_i of the node, one of the three affine transformations followed by a simple non-linearity is applied.

The dimensionality of the matrices and the bias vectors are determined by the dimensionalities $m^1, m^2, \ldots, m^D$ of representations at each level of the tree. The $W_{\mathtt{x}}^l$,$W_{\mathtt{y}}^l$, and $W_{\mathtt{z}}^l$ matrices at the lth level thus have the dimensionality $m^l \times 2m^{l+1}$ (recall that the levels are numbered from the root to the leaves) and the bias vectors $\mathbf{b}_{\mathtt{x}}^l, \mathbf{b}_{\mathtt{y}}^l, \mathbf{b}_{\mathtt{z}}^l$ have the dimensionality m^l.

Once the transformations (1) are applied in a bottom-up order, the root representation $\mathbf{v}_1(\mathcal{T})$ for the sample $\mathcal{T}$ is obtained. Naturally, it can be passed through several additional linear and non-linear transformations ("fully-connected layers"). In our classification experiments, we directly learn linear classifiers using $\mathbf{v}_1(\mathcal{T})$ representation as an input. In this case, the classification network output the vector of unnormalized class odds:

$$\mathbf{v}_0(\mathcal{T}) = W^0\mathbf{v}_1(\mathcal{T}) + \mathbf{b}^0\,, \tag{3}$$

where W^0 and $\mathbf{b}^0$ are the parameters of the final linear multi-class classifier.

3.3. Learning to classify

A Kd-network is a feed-forward neural network that has the learnable parameters $\{W_{\mathtt{x}}^j, W_{\mathtt{y}}^j, W_{\mathtt{z}}^j, \mathbf{b}_{\mathtt{x}}^j, \mathbf{b}_{\mathtt{y}}^j, \mathbf{b}_{\mathtt{z}}^j\}$ at each of the $D-1$ non-leaf levels $j \in \{1..D-1\}$, as well as the learnable parameters $\{W^0, \mathbf{b}^0\}$ for the final classifier. Standard backpropagation method can be used to compute the gradient of the loss function w.r.t. network parameters. The network parameters can thus be learned from the dataset of labeled kd-trees using standard stochastic optimization algorithms and standard losses, such as cross-entropy on the network outputs $\mathbf{v}_0(\mathcal{T})$ (3).

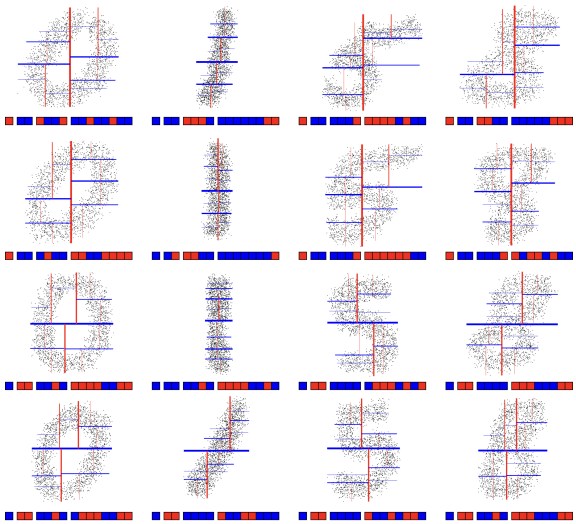

Figure 2. Kd-trees for MNIST clouds. We visualize several examples of 2D point clouds for MNIST (see text for description) with constructed kd-trees. The type of split is encoded with color and for each example the types of splits for the first four levels of the tree are shown below. Importantly, the structure of the kd-tree serves as a shape descriptor (e.g. 'ones' are dominated by vertical splits, and 'zeroes' tend to interleave vertical and horizontal splits as a kd-tree is traversed from the root to a leaf).

3.4. Learning to retrieve

It is straightforward to learn the representation (3) to produce not the class odds, but a descriptor vector of a certain dimensionality that characterizes the shape and can be used for retrieval. The parameters of the Kd-network can then be learned using backpropagation using any of the embedding-learning losses that observe examples of matching (e.g. same-class) and non-matching (e.g. different-class) shapes. In our experiments, we use a recently proposed histogram loss [33], but more traditional losses such as Siamese loss [5, 8] or triplet loss [27] could be used as well.

3.5. Properties of Kd-networks

Here we discuss the properties of the Kd-networks and also relate them to some of the properties of ConvNets.

Layerwise parameter sharing. Similarly to ConvNets, Kd-networks process the inputs by applying a sequence of parallel spatially-localized multiplicative operations interleaved with non-linearities. Importantly, just as ConvNets share their parameters for localized multiplications (convolution kernels) across different spatial locations, Kd-networks also share the multiplicative parameters $\{W_{\mathtt{x}}^j, W_{\mathtt{y}}^j, W_{\mathtt{z}}^j, \mathbf{b}_{\mathtt{x}}^j, \mathbf{b}_{\mathtt{y}}^j, \mathbf{b}_{\mathtt{z}}^j\}$ across all nodes at the tree level j.

Hierarchical representations. ConvNets apply bottom-up processing and compute a sequence of representations that correspond to progressively large parts of images. The

procedure is hierarchical, in the sense that a representation of a spatial location at a certain layer is obtained from the representations of multiple surrounding locations at the preceding layer using linear and non-linear operations. All this is mimicked in Kd-networks, the only difference being that the receptive fields of two different nodes at the same level of the kd-tree are non-overlapping.

Partial invariance to jitter. Convolutional networks that use pooling operations and/or strides larger than one are known to possess partial invariance to small spatial jitter in the input. Kd-networks are also invariant to such jitter (unless such jitter strongly perturbs the representations of leaf nodes). This is because the key forward-propagation operation (1) does ignore splitting thresholds τ_i. Thus, any small spatial perturbation of input points that leave the topology of the kd-tree intact can only affect the output of a Kd-network via the leaf representations (which as will be revealed in the experiments play only secondary role in kd-networks).

Non-invariance to rotations. Similarly to ConvNets, Kd-networks are not invariant to rotations, as the underlying kd-trees are not invariant to them. In this aspect, Kd-networks are inferior to intrinsic ConvNets [6, 2, 3]. Standard tricks to handle variable orientations include pre-alignment (using heuristics or network branches that predict geometric transformations of the data [13, 20]) as well as pooling over augmentations [14] (or simply training with excessive augmentations).

Role of kd-tree structure. The role of the underlying kd-trees in the process of Kd-network data processing is two-fold. Firstly, the underlying kd-tree determines which leaf representations are getting combined/merged together and in which order. Secondly, the structure of the underlying kd-tree can be regarded as a shape descriptor itself (Figure 2) and thus serves as the source of the information irrespective of what the leaf representations are. The Kd-network then serves as a mechanism for extracting the shape information contained in the kd-tree structure. As will be revealed in the experiments, the second aspect is of considerable importance, as even in the absence of meaningful leaf representations, Kd-networks are able to recognize shapes well solely based on the kd-tree structure.

3.6. Extension for segmentation

Kd-network architecture can be extended to perform semantic/part segmentation tasks in the same way as ConvNets. In this work, we mimic the encoder-decoder (hourglass-shaped) architecture with *skip connections* (Figure 3) that has been proposed for ConvNets in [17, 24]. More formally, during inference firstly the representations $\mathbf{v}_i$ are computed using (2), and then the second representation vector $\tilde{\mathbf{v}}_i$ is computed at each node i. The computations of the second representation proceed by setting $\tilde{\mathbf{v}}_1{=}\mathbf{v}_1$ (or obtaining $\tilde{\mathbf{v}}_1$ by one or several fully connected layers) and then using the following chain of top-down computations:

$$\tilde{\mathbf{v}}_{c_1(i)} = \phi([\tilde{W}^{l_i}_{d_{c_1(i)}}\tilde{\mathbf{v}}_i + \tilde{\mathbf{b}}^{l_i}_{d_{c_1(i)}}; S^{l_i}\mathbf{v}_{c_1(i)} + \mathbf{t}^{l_i}])\,,$$
$$\tilde{\mathbf{v}}_{c_2(i)} = \phi([\tilde{W}^{l_i}_{d_{c_2(i)}}\tilde{\mathbf{v}}_i + \tilde{\mathbf{b}}^{l_i}_{d_{c_2(i)}}; S^{l_i}\mathbf{v}_{c_2(i)} + \mathbf{t}^{l_i}])\,, \quad (4)$$

where $\tilde{W}^{l_i}_{d_{c_*(i)}}$ and $\tilde{\mathbf{b}}^{l_i}_{d_{c_*(i)}}$ are the parameters of the affine transformation that map the parent's representation to the children representations stacked on top of each other, while S^{l_i} and $\mathbf{t}^{l_i}$ are the parameters of the affine transformation within the skip connection from $\mathbf{v}_{c_1(i)}$ to $\tilde{\mathbf{v}}_{c_1(i)}$ (as well as from $\mathbf{v}_{c_2(i)}$ to $\tilde{\mathbf{v}}_{c_2(i)}$). In our implementation, the former set of parameters depends on split orientation, while the latter depends on the node layer only.

To increase the capacity of the model, additional multiplicative layers interleaved with non-linearities can be inserted in the beginning of the architecture or at the end of architecture (with parameters shared across leaves making these layers analogous to $1{\times}1$-convolutions in ConvNets). Also, fully-connected multiplicative layers can be inserted at the bottleneck.

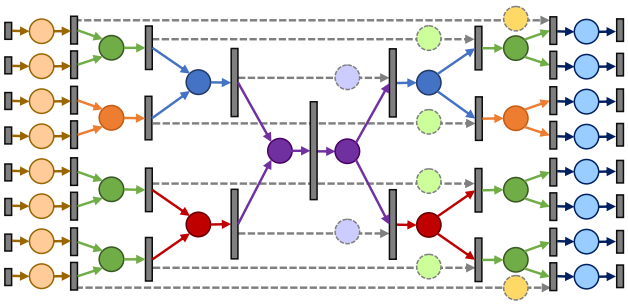

Figure 3. The architecture for parts segmentation (individual point classification) for the point cloud shown in Figure 1 (left). Arrows indicate computations that transform the representations (bars) of different nodes. Circles correspond to affine transformations followed by non-linearities. Similarly colored circles on top of each other share parameters. Dashed lines correspond to skip-connections (some "yellow" skip connections are not shown for clarity). The input representations are processed by an additional transformation (light-brown) and there are additional transformations applied to every leaf representation independently at the end of the architecture (light-blue).

3.7. Implementation details

Leaf representation. As mentioned above, for a leaf node i a representation $\mathbf{v}_i$ can be defined in several ways. In our experiments, unless stated otherwise, we use normalized 3D coordinates obtained by putting the center of mass of the shape at origin and rescaling the input point cloud to fit the $[-1; 1]^3$ 3D box.

Data augmentation. Similarly to other machine learning architectures, performance of Kd-networks can be improved through training data augmentations. Below, we

ModelNet	10-class		40-class	
Accuracy averaging	class	instance	class	instance
3DShapeNets [36]	83.5	-	77.3	-
MVCNN [31]	-	-	90.1	-
FusionNet [12]	-	93.1	-	90.8
VRN Single [4]	-	93.6	-	91.3
MVCNN [21]	-	-	89.7	92.0
PointNet [20]	-	-	86.2	89.2
OctNet [23]	90.1	90.9	83.8	86.5
ECC [29]	90.0	90.8	83.2	87.4
Kd-Net (depth 10)	92.8	93.3	86.3	90.6
Kd-Net (depth 15)	93.5	94.0	88.5	91.8
VRN Ensemble [4]	-	97.1	-	95.5
MVCNN-MultiRes [21]	-	-	91.4	93.8

Table 1. Classification results on ModelNet benchmarks. Comparison of accuracies of Kd-networks (depth 10 and 15) with state-of-the-art. Kd-networks outperform all single model architectures except MVCNNs, while performing worse than reported ensembles.

experiment with applying perturbing geometric transformations to 3D point clouds. Additionally, we found the injecting randomness into kd-tree construction very useful. For that, we randomize the choice of split directions using the following probabilities:

$$P\left(d_i = j|\hat{\mathbf{r}}_i\right) = \frac{\exp \gamma \hat{r}_i^j}{\sum_{j=x,y,z} \exp \gamma \hat{r}_i^j}\,, \tag{5}$$

where $\hat{r}_i$ is a vector of ranges normalized to unit sum.

4. Experiments

We now discuss the results of application of Kd-networks to shape classification, shape retrieval and part segmentation tasks benchmarks. For classification, we also evaluate several variations and ablations of Kd-networks. Our implementation of Kd-networks using Theano [32] and Lasagne [9] as well as additional qualitative and quantitative results are available at project webpage[1].

4.1. Shape classification

Datasets and data processing. We evaluate Kd-networks on datasets of 2D (for illustration purposes) as well as 3D point clouds. 2D point clouds were produced from the **MNIST** dataset [15] by turning centers of non-zero pixels into 2D points. A point cloud of a needed size was then sampled from the resulting set of points with an addition of a small random noise. Figure 2 shows examples of resulting point clouds.

The 10-class and the 40-class variations of ModelNet [36] (**ModelNet10** and **ModelNet40**) benchmarks, containing 4899 and 12311 models respectively, were used for 3D shape classifications. The two datasets are split into the training set (3991 and 9843 models) and the test set (909 and 2468 models respectively). In this case, 3D point clouds were computed as follows: firstly, a given number of faces were sampled with the probability proportionate to their surface areas. Then, for the sampled face a random point was taken. The whole sampling procedure thus closely approximated uniform sampling of model surfaces.

[1] http://sites.skoltech.ru/compvision/kdnets/

	MNIST	ModelNet10	ModelNet40
Split-based linear	82.4	83.4	73.2
Kd-net RT+SA (no leaf)	98.6	92.7	89.8
Kd-net DT	98.9	89.2	85.7
Kd-net RT	99.1	92.8	89.9
Kd-net RT+TA	99.1	92.9	90.1
Kd-net RT+SA	99.1	93.2	90.6
Kd-net RT+SA+TA	99.1	93.3	90.6

Table 2. Classification accuracy for baselines and different data augmentations. The resulting accuracies for the baseline model, the ablated model with trivial leaf representations, as well as Kd-networks trained with various data augmentations. DT = deterministic kd-trees, RT = randomized kd-trees, TA = translation augmentation, SA = anisotropic scaling augmentation. All networks are depth 10. See text for discussion.

Training and test procedures. Additionally we preprocess each object by applying a geometric perturbation and noise (as discussed below). Either a deterministic or a randomized kd-tree is constructed and, finally, the resulting point cloud and leaf representations are used to perform forward-backward pass in the Kd-Network. At test time, we use the same augmentations as were used during training and average predicted class probabilities over ten runs.

We experimented with the following augmentations: (i) proportional translations along every axis (*TR*) of up to ± 0.1 in normalized coordinates; proportional anisotropic rescaling over the two horizontal axes (*AS*) by the number sampled from the 0.66 to 1.5 range. More global augmentations like flips or rotations did not improve results. Additionally, we evaluated both deterministic (DT) and randomized (RT) kd-trees. For our experiments we fixed the parameter γ in (5) to ten.

Benchmarking classification performance. We compare our approach to the state-of-the-art on the ModelNet10 and ModelNet40 benchmarks in Table 1. We give the results obtained with kd-trees of depth 10 and depth 15. For depth 10, our architecture firstly obtains leaf representation of size 32 from initial points coordinates with an affine transformation with parameters shared across all the input points interleaved with a ReLU non-linearity, then a *Kd-network* obtains intermediate representations of sizes: $32 - 64 - 64 - 128 - 128 - 256 - 256 - 512 - 512 - 128$. Resulting representation for a point cloud is directly used to obtain class posteriors with a single fully connected layer.

For depth 15, the previous architecture has been modified by changing the size of leaf representation to 8 and by updated progression of intermediate representation sizes: $16-16-32-32-64-64-128-128-256-256-512-512-1024-1024-128$.

In both cases, we used translation-based and anisotropic scaling-based augmentations as well as randomized kd-tree generation at test and at train time. Note that despite the use of random augmentations, a single model (i.e. a single set of model weigths) was evaluated for each of the cases (depth 10 and depth 15). Our results are better than all previous single-model results on these benchmarks except MVCNNs. While being worse than the reported ensembles, Kd-networks can be trained faster. VRN ensemble involves 6 models each trained over the course of 6 days on NVidia Titan X. Our depth-10 model can be trained in 16 hours, and our depth-15 model can be trained in 5 days using an older NVidia Titan Black. Furthermore, more than 75% of the time is spent on point cloud sampling and kd-tree fitting, while the training itself takes less then a quarter of the mentioned times.

It is also interesting to note that the performance of Kd-networks on the MNIST dataset reaches 99.1% (Table 2), which is in the ballpark of the results obtained with ConvNets (without additional tricks).

Ablations and variants. Kd-networks use two sources of information about each object, namely the leaf representations and the direction of the splits. Note, that the split coordinates are not used in the classification. We assess the relative importance of the two sources of the information using two baselines. Firstly, we consider the baseline for both 2D and 3D point clouds that encode split information from their kd-trees in the following way: every split on every level is one-hot encoded and concatenated to resulting feature vector. We then use a linear classifier on such a representation (which is also shown as red/blue bars in Figure 2). This baseline evaluates how much information can be recovered from the split orientation information with very little effort.

We also evaluate a model ablation corresponding to our full method with the exception that we remove the first source information. To this end, we make each leaf representation equal a one-dimensional vector (i.e. scalar) that equals one, effectively removing the first source of information.

The results in Table 2 suggest that the first (linear classification) baseline performs much worse than Kd-network (even without leaf information), which suggests that multi-stage hierarchical data flow and intricate weight sharing mechanism of Kd-networks plays an important role (note, however that this baseline performs considerably better than chance suggesting that the orientation of splits in a kd-tree can serve as shape descriptor). Most interestingly, the ablated version of Kd-network comes very close to the full method, highlighting that the second source of information (split direction) dominates the first in terms of importance (confirming the suitability of kd-trees for shape description).

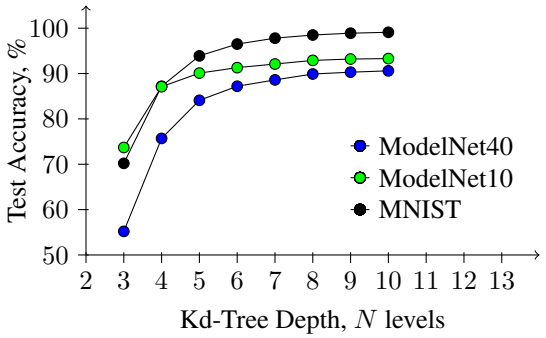

Figure 4. Kd-tree depth experiments. Test accuracy for Kd-networks trained on clouds of different size 2^N (corresponding to kd-tree depth N). Saturation without overfitting can be observed.

Finally, in Table 2 we assess the importance of two different augmentations as well as the relative performance of randomized and deterministic trees. These experiments suggest that the randomization of kd-tree boosts the performance (generalization) considerably, while the geometric augmentations give a smaller effect.

Kd-tree depth experiments. For better understanding of the effect of depth, we also conducted a series of experiments corresponding to trees of different depths (Figure 4) of less or equal than ten. To obtain Kd-network architectures for smaller depths we simply remove initial layers from our 10-depth architecture (described above).

Apart from the saturating performance, we observe that the learning time for each epoch for smaller models becomes very short but the number of epochs to achieve convergence increases. For bigger models the time of kd-tree construction (and point sampling) becomes the bottleneck in our implementation.

Degradation in the presence of non-uniform sampling and jitter. We have also measured the degradation of Kd-networks in the presence of non-uniform sampling and jitter and provide the results in the supplementary material. Overall, degradation from both effects on the ModelNet10 benchmark is surprisingly graceful.

4.2. Shape retrieval

Dataset and data processing. For the purpose of evaluation for 3D shape retrieval task we use ShapeNetCore dataset [7]. ShapeNetCore is a subset of full ShapeNet dataset of 3D shapes with manually verified category annotations and alignment. It consists of 51300 unique 3D shapes divided into 55 categories each represented by its triangular meshes. For our experiments we used a distri-

	Micro					Macro				
	P@N	R@N	F1@N	mAP	NDCG@N	P@N	R@N	F1@N	mAP	NDCG@N
Bai [26]	0.706	0.695	0.689	0.825	0.896	0.444	0.531	0.454	0.740	0.850
Su [26]	0.770	0.770	0.764	0.873	0.899	0.571	0.625	0.575	0.817	0.880
Kd-net (depth 15)	0.760	0.768	0.743	0.850	0.905	0.492	0.676	0.519	0.746	0.864
Bai [26]	0.678	0.667	0.661	0.811	0.889	0.414	0.496	0.423	0.730	0.843
Su [26]	0.632	0.613	0.612	0.734	0.843	0.405	0.484	0.416	0.662	0.793
Kd-net (depth 15)	0.473	0.519	0.451	0.617	0.814	0.205	0.529	0.241	0.484	0.726
MVKd-net (depth 10)	0.660	0.652	0.631	0.766	0.868	0.355	0.560	0.382	0.617	0.792

Table 3. Retrieval results on normal and perturbed (top and bottom respectively) version of ShapeNetCore dataset for the metrics introduced in [26] (higher is better). See [26] for the details of metric and the presented systems (in general, all systems in [26] incorporated some variants of 2D multi-view ConvNets). Kd-networks perform on par with the system of Su et al. that is based on multi-view ConvNet[31] and generally better than other methods in case of pose normalized dataset. For the perturbed version of the dataset, Kd-network suffer from degradation in performance due to sensitivity to global rotation. Multi-view (20 random views) version of Kd-network again perform on par with most sophisticated multi-view ConvNets.

bution of the dataset and a training/validation/test split provided by the organizers of 3D Shape Retrieval Contest 2016 (SHREC16) [26]. Apart from the aligned shapes this distribution contains a perturbed version of the dataset, which consists of the same shapes each perturbed by a random rotation. Also, there is an additional division into several subcategories available for each category. In our experiments we evaluate on both versions of the dataset.

Training and test procedures. We used a two stage training procedure for the object retrieval task. Firstly, the network was trained to perform classification task in the manner described above. Secondly, the final layer of the network predicting the class posterior was removed, resulting representations of point clouds were normalized and used as shape descriptors provided for the fine-tuning of the network with histogram loss. A mini-batch of size 110 was used for training, each containing two randomly selected shapes from each category of the dataset. Both training and prediction was done with geometric perturbations and kd-tree randomization applied. The parameters of the augmentations were taken from the classification task. To improve stability and quality of prediction at test time for each model the descriptors were averaged over several (16 in this experiment) randomized kd-trees before normalization.

Benchmarking retrieval performance. We compare our results Table 3 with the results of the participants of SHREC'16 for both normal and perturbed versions of ShapeNetCore. Most participating teams of SHREC'16 challenge used systems based on multi-view 2D ConvNets. We use the metrics introduced in [26]. Macro averaged metrics are computed by simple averaging of a metric across all shape categories, micro averaged metrics are computed by weighted averaging with weights proportionate to the number of shapes in a category. A depth-15 Kd-network trained with the histogram loss [33] was used for this task with leaf representation of size 16 (obtained from the three coordinates using an additional multiplicative layer) and intermediate representations of sizes $32-32-64-64-128-128-256-256-512-512-1024-1024-2048-2048-512$. The obtained descriptors of size 512 were used to compute similarity and make predictions for each shape. A similarity cutoff was chosen from the results obtained on the validation part of the datasets.

In general our method performs on par with the system based on multiview CNNs [31], and better than other systems that participated in SHREC'16 for the 'normal' set. For the 'perturbed' version, the performance of Kd-networks suffers from non-invariance to global rotations. To address this, we implemented a simple modification (in the spirit of the TI-Pooling [14]) that applies Kd-network (depth 10) to 20 different random rotations of a model and performs max-pooling over the produced representations followed by three fully connected layers to produce final shape descriptors. The resulting system achieved a competitive performance on the 'perturbed' version of the benchmark (Table 3).

4.3. Part Segmentation

Finally, we used the architecture discussed in Section 3.6 to predict part labels for individual points within point clouds (e.g. in an airplane each point can correspond to body, wings, tail or engine).

Dataset and data processing. We evaluate our architecture for part segmentation on ShapeNet-part dataset from [37]. It contains 16881 shapes represented as separate point clouds from 16 categories with per point annotation (with 50 parts in total). In this dataset, both the categories and the parts within the categories are highly imbalanced, which poses a challenge to all methods including ours.

Training and test procedures. Since the number of points representing each model differs in the dataset, we upsample each point cloud to size 4096 by duplicating random

	mean	aero plane	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor bike	mug	pistol	rocket	skate board	table
Yi [37]	81.4	81.0	78.4	77.7	75.7	87.6	61.9	92.0	85.4	82.5	95.7	70.6	91.9	85.9	53.1	69.8	75.3
3DCNN [20]	79.4	75.1	72.8	73.3	70.0	87.2	63.5	88.4	79.6	74.4	93.9	58.7	91.8	76.4	51.2	65.3	77.1
PointNet [20]	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
Kd-network	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3

Table 4. Part segmentation results on ShapeNet-core dataset. The Intersection-over-Union scores are presented for each category as well as mean IoU are reported. Kd-network do not outperform PointNet, although for some classes the performance of Kd-networks is competitive or better.

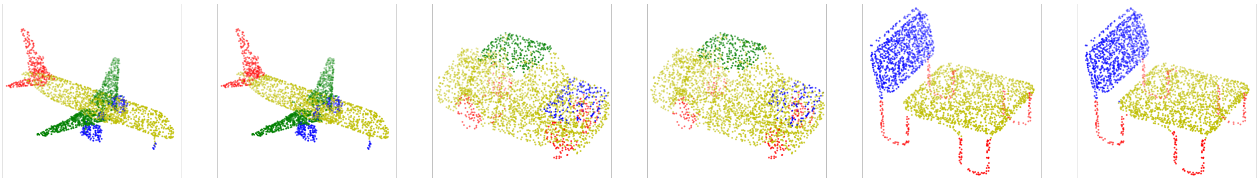

Figure 5. Examples of part segmentation resulting point labeling (use zoom-in for better viewing). Each pair of shapes contain ground truth labeling on the left and predicted labeling on the right. The examples were randomly taken from the validation part of the ShapeNet-core dataset.

points with an addition of a small noise. Apart from making data feasible for our method, such upsampling helps with rare classes. The upsampled point clouds then are fed to the architecture shown in the Figure 3, which is optimized with the mean cross entropy over all points in a cloud as a loss function. During test time predictions are computed for the upsampled clouds, then the original cloud is passed through a constructed kd-tree to obtain a mapping of each leaf index to corresponding set of original points. This is further used to produce final predictions for every point. Similar to other tasks, we have used data augmentations both during training and test times and averaged predictions over multiple kd-trees.

Benchmarking part segmentation performance. Our results are compared to 3D-CNN (reproduced from [20]), PointNet architecture [20], and the architecture of [37]. For each category mean intersection over union (IoU) is considered as a metric: for each shape IoUs are computed as an average of IoUs for each part which is possible to occur in this shape's category. Resulting shape IoUs are averaged over all the shapes in the category. A depth 12 variant of Kd-network was used for this task with leaf representations of size 128 and intermediate representations of sizes $128-128-128-256-256-256-256-512-512-512-512-1024$. Two additional fully connected layers of sizes 512 and 1024 was used in the bottleneck of the architecture. The output of segmentation network is further processed by three affine transformations interleaved with ReLU non-linearities of sizes 512, 256, 128. The probabilities of the 50 parts present in all classes in the dataset are predicted (the probabilities of the parts that are not possible for a given class are ignored following the protocol of [20]). Batch-normalization is applied to each layer of the whole architecture.

The performance of Kd-networks (Table 4) for the part segmentation task is competitive though not improving over state-of-the-art. We speculate that one of the reasons could be insufficient propagation of information across high-level splits within kd-tree, although resulting segmentations do not usually show the signs of underlying kd-tree structure (Figure 5). A big advantage of Kd-networks for the segmentation task is their low memory footprint. Thus, for our particular architecture, the footprint of one example during learning is less than 120 Mb.

5. Conclusion

In this work we propose new deep learning architecture capable of production of representations suitable for different 3D data recognition tasks which works directly with point clouds. Our architecture has many similarities with convolutional networks, however it uses kd-tree rather than uniform grids to build the computational graphs and to share learnable parameters. With our models we achieve results comparable to current state-of-the-art for a variety of recognition problems. Compared to the top-performing convolutional architectures, kd-trees are also efficient at test-time and train-time.

The competitive performance of our deep architecture based on kd-trees suggests that other hierarchical 3D space partition structures, such as octrees, PCA-trees, bounding volume hierarchies ould be investigated as underlying structures for deep architectures.

Acknowledgement: this work is supported by the Russian MES grant RFMEFI61516X0003.

References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[2] D. Boscaini, J. Masci, S. Melzi, M. M. Bronstein, U. Castellani, and P. Vandergheynst. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. *Comput. Graph. Forum*, 34(5):13–23, 2015.

[3] D. Boscaini, J. Masci, E. Rodolà, and M. M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Proc. NIPS*, pages 3189–3197, 2016.

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

[5] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah. Signature verification using a siamese time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688, 1993.

[6] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[7] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[8] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proc. CVPR*, pages 539–546, 2005.

[9] S. Dieleman, J. Schlter, C. Raffel, E. Olson, et al. Lasagne: First release., Aug. 2015.

[10] J. D. Foley, A. Van Dam, S. K. Feiner, J. F. Hughes, and R. L. Phillips. *Introduction to computer graphics*, volume 55. Addison-Wesley Reading, 1994.

[11] A. Guttman, M. Stonebraker, and C. U. B. E. R. LAB. *R-trees: A Dynamic Index Structure for Spatial Searching*. Memorandum (University of California, Berkeley, Electronics Research Laboratory). Defense Technical Information Center, 1983.

[12] V. Hegde and R. Zadeh. Fusionnet: 3d object classification using multiple data representations. *arXiv preprint arXiv:1607.05695*, 2016.

[13] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *Proc. NIPS*, pages 2017–2025, 2015.

[14] D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys. TI-POOLING: transformation-invariant pooling for feature learning in convolutional neural networks. In *Proc. CVPR*, pages 289–297, 2016.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[16] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas. Fpnn: Field probing neural networks for 3d data. In *Proc. NIPS*, 2016.

[17] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proc. CVPR*, pages 3431–3440, 2015.

[18] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Proc. IROS*, pages 922–928. IEEE, 2015.

[19] D. J. Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980.

[20] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[21] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proc. CVPR*, 2016.

[22] A. Requicha, H. Voelcker, and U. of Rochester. Production Automation Project. *Constructive Solid Geometry*. TM (Rochester, PAP). Production Automation Project, University of Rochester, 1977.

[23] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[24] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proc. MICCAI*, pages 234–241. Springer, 2015.

[25] H. Samet. *The design and analysis of spatial data structures*, volume 199. Addison-Wesley Reading, MA, 1990.

[26] M. Savva, F. Yu, H. Su, M. Aono, B. Chen, D. Cohen-Or, W. Deng, H. Su, S. Bai, X. Bai, et al. SHREC16 track large-scale 3d shape retrieval from ShapeNet Core-55. In *Proceedings of the Eurographics Workshop on 3D Object Retrieval*, 2016.

[27] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. *Advances in neural information processing systems (NIPS)*, page 41, 2004.

[28] R. A. Schumacker, B. Brand, M. G. Gilliland, and W. H. Sharp. Study for applying computer-generated images to visual simulation. Technical report, DTIC Document, 1969.

[29] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proc. CVPR*, 2017.

[30] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proc. ICML*, pages 129–136, 2011.

[31] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, pages 945–953, 2015.

[32] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[33] E. Ustinova and V. S. Lempitsky. Learning deep embeddings with histogram loss. In *Proc. NIPS*, pages 4170–4178, 2016.

[34] D. Z. Wang and I. Posner. Voting for voting in online point cloud object detection. In *Proc. RSS*, 2015.

[35] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Proc. NIPS*, pages 82–90, 2016.

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

[37] L. Yi, V. G. Kim, D. Ceylan, I. Shen, M. Yan, H. Su, A. Lu, Q. Huang, A. Sheffer, L. Guibas, et al. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (TOG)*, 35(6):210, 2016.