PointTree: Transformation-Robust Point Cloud Encoder with Relaxed K-D Trees

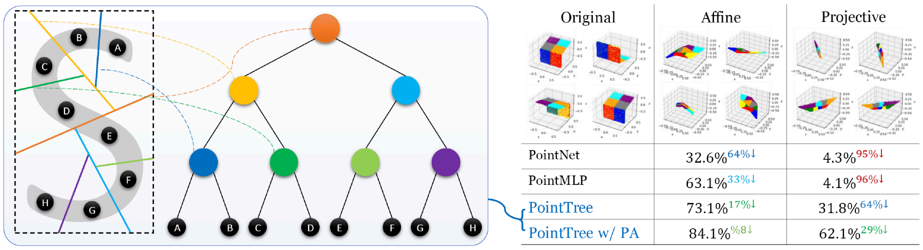
Jun-Kun Chen and Yu-Xiong Wang

University of Illinois at Urbana-Champaign
{junkun3, yxw}@illinois.edu

Abstract. Being able to learn an effective semantic representation directly on raw point clouds has become a central topic in 3D understanding. Despite rapid progress, state-of-the-art encoders are restrictive to canonicalized point clouds, and have weaker than necessary performance when encountering geometric transformation distortions. To overcome this challenge, we propose *PointTree*, a general-purpose point cloud encoder that is *robust to transformations* based on *relaxed* K-D trees. Key to our approach is the design of the division rule in K-D trees by using principal component analysis (PCA). We use the structure of the relaxed K-D tree as our computational graph, and model the features as border descriptors which are merged with pointwise-maximum operation. In addition to this novel architecture design, we further improve the robustness by introducing *pre-alignment* – a simple yet effective PCA-based normalization scheme. Our PointTree encoder combined with pre-alignment consistently outperforms state-of-the-art methods by large margins, for applications from object classification to semantic segmentation on various transformed versions of the widely-benchmarked datasets. Code and pre-trained models are available at https://github.com/immortalCO/PointTree.

1 Introduction

3D sensing technology has advanced rapidly over the past few years, playing a significant role in many applications such as augmented reality, autonomous driving, and geographic information systems [1,6,9]. As one of the most commonly-used output formats of 3D sensors, 3D point clouds flexibly describe the surface information of the sensed objects or scenes with collection of points. Therefore, being able to learn an effective semantic representation directly on raw point clouds, which is useful for high-level tasks such as object recognition, has become a central topic in 3D understanding, with various powerful deep learning architecture based encoders emerging like PointNet [18] and PointMLP [17]. In real-world applications, a desired encoder is supposed to cope with a wide range of *geometric transformations* – point clouds of the same object category may undergo different similarity/affine/projective transformations, leading to



PointNet	32.6%$^{64\%\downarrow}$	4.3%$^{95\%\downarrow}$
PointMLP	63.1%$^{33\%\downarrow}$	4.1%$^{96\%\downarrow}$
PointTree	73.1%$^{17\%\downarrow}$	31.8%$^{64\%\downarrow}$
PointTree w/ PA	84.1%$^{\%8\downarrow}$	62.1%$^{29\%\downarrow}$

(a) (b)

Fig. 1: Our **robust** PointTree vs. **non-robust** existing point cloud encoders under geometric transformations. **(a)** PointTree is based on *relaxed* K-D trees, and its robustness is mainly achieved by designing the division plane of each node to be robust against transformations: An example of a relaxed K-D tree operated on an 8-point S-shaped point cloud. Each node is related to the division line with the same color, and each point is related to a leaf node. **(b)** Affine and projective transformations can highly deform the point clouds, dramatically decreasing the object classification accuracy of existing point cloud encoders like the widely-used PointNet [18] and more recent PointMLP [17] ('↓' indicates the relative performance drop compared with their accuracy on the canonicalized point clouds which is 90.6% and 94.5%, respectively). We illustrate an example point cloud in ModelNet40 [27]. For the original (canonicalized), affine, and projective versions, we show 4 different point of views of the same point cloud, and use different colors to represent different octants. PointTree significantly outperforms its counterparts in challenging transformation scenarios. With the additionally proposed pre-alignment ('PA'), PointTree further improves the accuracy

large intra-class variations. This paper demonstrates that existing state-of-the-art point cloud encoders have weaker than necessary performance when encountering geometric transformation distortions, and proposes *PointTree*, a new type of encoder based on *relaxed* K-D trees that is *robust to transformations* and thus offers substantial improvements in performance.

Such investigation on transformation robustness for point cloud encoders is under-explored in existing work, partially because most of the commonly-used benchmark datasets, like ModelNet [27] and ShapeNet [3], make a simplifying assumption – the point clouds are in precise shapes and aligned in a canonical coordinate system. For example, a point cloud of a table in such a dataset always has its tabletop parallel to the xOy plane, and all its legs vertical to the top. However, such an assumption is restrictive and hardly satisfied *in the wild* – e.g., an object scanned by a 3D sensor is often not aligned with the sensor, leading to unaligned point clouds. Due to noise, perturbation, viewpoint change, miscalibration, or precision limitation of the sensor itself [16,24,34], the scanned point cloud may have a different shape from the object, and the shape might be deformed by a 3D affine or projective transformation. On the other hand, most of existing encoders developed on these benchmarks highly rely on this assumption and are thus sensitive to input point cloud deformations, leading to dramatically degenerated recognition performance as shown in Figure 1-b. Notably, while effective for simple input corruptions, the common strategy (e.g., the T-Net [18] used in PointNet) that adopts a transformer network to explicitly predict a transformation for canonicalization of input data cannot deal with more general deformations here.

To overcome this challenge, we propose PointTree, a transformation-robust, general-purpose point cloud encoder architecture. Key to our approach is the use of *relaxed* K-D trees [4]. While there have been some existing approaches [10,33] that utilize K-D trees, they are based on the *conventional* K-D tree that only divides the point set along an axis at each node, which implicitly uses the aligned assumption and thus still performs poorly on transformed point clouds. By contrast, our use of the relaxed K-D tree removes the restriction of division rules, allowing more flexible designs. Particularly, in PointTree, we design the division rule by using principal component analysis (PCA) as illustrated in Figure 1-a. By doing so, theoretically, we show that our division of the point set and construction of the whole K-D tree are *invariant to similarity transformations*; and empirically, we observe that our PointTree exhibits strong *robustness against more complicated affine and projective transformations* (Figure 1-b).

In addition to the proposed division rule, the robustness of our approach further stems from other properties of K-D trees and additional design strategies. As a tree structure containing multiple layers, PointTree natively divides a point cloud into components at bottom layers. This facilitates the recognition of multi-component objects (e.g., an airplane), as PointTree may still capture useful local features from lower layers even if the whole point cloud undergoes severe deformation. Also, the similarity transformation-invariant division induced by relaxed K-D trees prevents cutting two components with the same shape (e.g., two engines of the airplane) in different directions, so that the symmetricity can still be leveraged. Moreover, following PointNet [18] and PointNet++ [19], we model the features in PointTree as border descriptors which can be merged with pointwise-maximum operation. We use the structure of the relaxed K-D tree as our computational graph, which contains a native locality clustering and down-sampling scheme. Finally, not only from this novel architecture design, but we also improve the robustness by introducing a simple PCA-based normalization scheme (called "pre-alignment") on input point clouds to PointTree. This is shown as a general normalization scheme that consistently and effectively improves the performance of other encoders under transformations as well.

Our contributions are three-folds. (1) We propose PointTree, a general-purpose point cloud encoder architecture based on relaxed K-D trees, which is robust against geometric (affine and projective) transformations. (2) We introduce pre-alignment, a simple yet general PCA-based normalization scheme, which can consistently improve the performance of a variety of point cloud encoders under geometric transformations. (3) We show that our PointTree encoder combined with pre-alignment consistently outperforms state-of-the-art methods by large margins, on various transformed versions of ModelNet40 [27], ShapeNet-Part [3], and S3DIS [2] benchmarks.

2 Related Work

Deep Learning on Point Clouds. There are mainly four directions to build a deep learning model to process and analyze point clouds [7]: (i) multi-layer perceptron (MLP) methods [17–19] that use pointwise MLPs along with some multi-stage locality clustering and down-sampling; (ii) convolution methods [14, 15] that perform convolutions on voxels, grids, or directly in continuous 3D space; (iii) graph methods [26,35] that construct graphs with points as vertices and with neighborhood relations as edges, and apply graph models; and (iv) data structure methods [10,21,22,33] that use a hierarchical data structure like an OCTree or a K-D tree as the computational graph. Our PointTree belongs to a data structure method, since it uses a relaxed K-D tree as the computational graph. Intuitively, pure MLP methods (without locality clustering) are not robust against transformations, since they only rely on coordinate values; by exploiting locality, locality clustering and neighborhood graph may improve the robustness.

Point Cloud Encoders Based on K-D Trees. To the best of our knowledge, mainly four methods in the literature use K-D trees to build point cloud encoders: KD-Net [10], 3DContextNet [33], PD-Net [31], and MRT-Net [5], while more approaches exist based on OCTrees [11,21,22,28]. KD-Net is a simple version of the K-D tree-based point cloud encoder – it uses MLP to merge the information of two children nodes for each node. As an advanced version, 3DContextNet uses the border descriptor features from PointNet [18] which can be merged by pointwise-maximum; it further proposes multi-stage training to exploit local and global context. PD-Net [31] is a variant of KD-Net that replaces the vanilla K-D tree with a PCA-based K-D tree, but its feature aggregation still highly relies on coordinates. MRT-Net [5] uses the K-D tree only for preprocessing and uses convolutional layers for further modeling.

Relation Between Our PointTree and Previous Models. Instead of using conventional K-D trees, PointTree leverages relaxed K-D trees with a proposed division plane selection method, making it different from all existing K-D tree-based models. Similar to PointNet [18] and 3DContextNet [33], PointTree models the features as border descriptors. While the model design of PointTree is inspired by and similar to that of 3DContextNet in the "feature learning stage," PointTree is simpler *without* relying on multi-stage training and local and global cues. Furthermore, PointTree has an extra alignment network as in PointNet.

Investigation on Transformation Robustness. While there has been interest in addressing robustness against geometric transformations of input point clouds, existing work mainly focuses on specific transformations like rotation and similarity. IT-Net [32] proposes a learnable normalization component (or "alignment network") which learns to recover the original point cloud. Other work [12, 13, 23, 36] proposes SO(3) (similarity transformation) robust, invariant, or equivariant architectures that maintain stable results when training on rotated point clouds. Shear transformation is also studied [25], which is a special type of affine transformation with deformation only performed on two of the three axes. None of the existing methods are able to cope with robustness against general affine or projective transformations as our work.

3 Methodology

Problem Setting. We design a deep learning model as a general-purpose point cloud encoder. A **point cloud** is an unordered set of points $P = \{p_i\}_{i=1}^n$. Each point p_i is a 3-D vector (x_i, y_i, z_i) representing the three coordinates. Our model directly takes P as input, and outputs a set of vectors O. For a downstream task, another model takes O as input, and outputs task-specific predictions.

3.1 Point Cloud Encoder Based on Relaxed K-D Trees

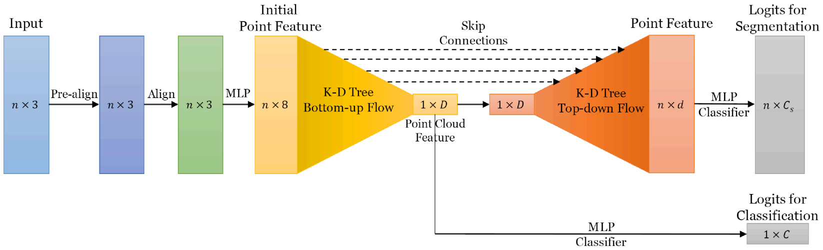


Fig. 2: Architecture of PointTree. The input point cloud passes through a pre-alignment process, followed by an alignment network, and is then fed to our PointTree model. A K-D tree bottom-up flow is applied on such input, obtaining the point cloud feature which can be used for classification. For segmentation tasks, another K-D tree top-down flow is applied on the output of the bottom-up flow with some skip connections, obtaining point features which can be used for segmentation

K-D Trees and Relaxed K-D Trees. Our proposed point cloud encoder *PointTree* is based on relaxed K-D trees, as illustrated in Figure 2. K-D trees are a classical data structure designed to solve K-dimensional range counting problems. It is a special decision tree built on n K-dimensional input points, in two specific ways: (i) each node has an axis-parallel criterion; and (ii) such a criterion strictly divides the input points that go through this node to two equal-size parts. Each leaf node is related to exactly one input point.

A K-D tree of depth d is a full binary tree with 2^d points. The root is at layer 0; the leaves are at layer d. Each non-leaf node o has two unordered children nodes o_l and o_r, while each leaf node has a corresponding point $p(o)$. Each non-root node has a unique parent node $\mathbf{par}(o)$. At each non-leaf node, a linear criterion $W_o p + b_o \leq 0$ divides the point set into two subsets, which is recursively processed at the left and right nodes. Hence, the subtree of each node contains the points in a continuous 3-D space, leading to a native locality clustering.

Existing K-D tree-based methods [10, 33] use the conventional definition of K-D trees that only divides along one axis, i.e., $W_o \in \{(0, 0, 1), (0, 1, 0), (1, 0, 0)\}$.

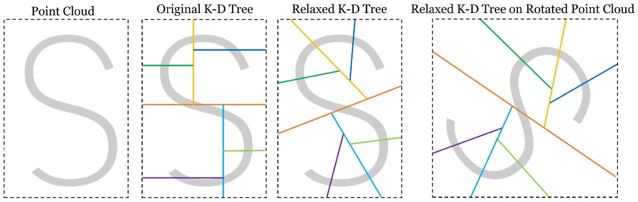


Fig. 3: Relaxed K-D tree's **division and similarity equivariance**. For the S-shape point cloud, the original K-D tree divides it into a large number of *imbalanced* segments (one of the parts even contains two disconnected segments of S), while the relaxed K-D tree can cut it into a small number of *balanced* segments. Also, when rotating the S-shape with an angle, each division plane and pieces of the point cloud divided by the relaxed K-D tree rotate with the same angle and each divided segment keeps unchanged. This shows that relaxed K-D trees are equivariant to similarity transformations

However, such methods are not exploiting the expressiveness of K-D trees. By contrast, we adopt **relaxed K-D trees** [4] – a generalization of K-D trees by removing the restriction of division plane, so that the point set can be divided with any criterion. Here, to improve the transformation robustness, we consider an arbitrary linear criterion, i.e., W_o can be an arbitrary vector.

Concretely, PointTree uses the first principle component (obtained by a PCA algorithm) as the division plane, and chooses the medium value as the division boundary. As PCA is similarity-transformation equivariant, this K-D tree construction has the invariance against similarity transformations, as shown in Figure 3. Importantly, this property also brings in strong robustness against affine and projective transformations, as empirically validated in Section 4. In addition, if the point cloud contains multiple similar components due to repeating or symmetricity, e.g., airplane engines, table legs, and desk drawers, our similarity-transformation invariant K-D tree construction can divide these components in the same way, further improving the model robustness and facilitating feature extraction from these components.

Bottom-Up Information Flow. In a traditional K-D tree algorithm, there is a scheme to upload information in a bottom-up way, called **bottom-up information flow**. In such information flow, each node o takes some information as a vector info(o). The information at each leaf node is derived from the corresponding point, and the information at each non-leaf node is the aggregated information of its children nodes, as the formula below:

$$\text{info}(o) = \begin{cases} \textbf{point-info}_o(p(o)), & o \in L_d, \\ \textbf{merge-info}_o(\text{info}(o_l), \text{info}(o_r)), & o \in L_i. \end{cases} \tag{1}$$

The uploading process of the information flow is a natural down-sampling scheme. In the layer-by-layer bottom-up uploading process, the number of nodes in layer i (which is 2^i) is decreasing, and the size of the subtree of each node (which contains information of 2^{d-i} points) is increasing. Thus, we can regard

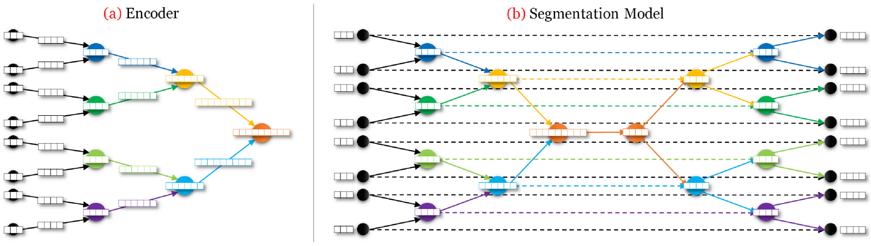


Fig. 4: (a) Our PointTree encoder. The encoder has a multi-stage down-sampling and dimension increasing scheme, where each layer in the K-D tree is a stage. The points are down-sampled through layers via a bottom-up information flow. The grids on the nodes and edges show the dimension of features. For each node, the features from its two children nodes are first fed to a dimension-increasing MLP, and then merged with pointwise-maximum operation. **(b) The segmentation model**. The two symmetric K-D trees represent two stages: a bottom-up information flow (**left, the same as (a) with simplified notation**) and a top-down information flow (**right**), on the same K-D tree. The information is first uploaded from leaves to root to obtain global features, and then downloaded from root to leaves, obtaining the relationship between each leaf/point and the whole point cloud which is used for segmentation classification. For the top-down information flow (**right**), each node has a carried feature that represents the information of the relationship between its subtree and the whole point cloud. Such feature is obtained by merging its parent nodes' carried information and with a skip connection (dotted lines) from the same node in the bottom-up feature

each layer as a stage of down-sampling with down-sampled points. Such down-sampling process starts with the information of original points P and ends with a single point info(R), where R is the root of the K-D tree.

Our Encoder. We design the encoder of PointTree (see Figure 4-a) by leveraging the idea of PointNet [18] and its improved version PointNet++ [19], but in the information flow of a relaxed K-D tree. Specifically, we model the information as the border descriptor or "global shape descriptor" defined in PointNet [18], which can be aggregated with pointwise-maximum operation (a.k.a. maximum pooling). For each node o, the information or feature info(o) is the coarse shape descriptor of the points in its subtree.

We compute the information of each node layer-by-layer in a bottom-up order. For leaf nodes, the information is obtained by applying an MLP on the coordinates. For non-leaf nodes, the information is obtained by processing their left and right children nodes with a dimension-increasing linear transformation (defined on each layer), followed by aggregation with a pointwise-maximum operation, as the formula below:

$$\text{info}(o) = \begin{cases} \mathbf{MLP}(p(o)), & o \in L_d, \\ \textbf{pointwise-max}(W_i \text{info}(o_l), W_i \text{info}(o_r)), & \text{otherwise}. \end{cases} \tag{2}$$

The encoder of PointTree can be regarded as a K-D tree-guided version of PointNet++, where at each stage of down-sampling, the points are clustered under some rule, and each cluster is down-sampled to one point with higher dimension. In PointTree, instead of some ad hoc clustering strategies used by PointNet++, K-D trees provide a native and principled way to cluster according to the spatial and neighborhood information, making it more powerful, general, and reliable. The output of the PointTree encoder is defined as $O = \{\text{info}(o) \mid o\}$.

3.2 Robustness Against Transformations

The robustness of PointTree against transformations mainly stems from our design of the division rule. Here, we discuss the robustness in more detail, introduce additional strategies that further improve the robustness, and propose a metric that quantifies the transformation intensity.

Similarity Transformation. PointTree uses relaxed K-D trees as its base tree, which holds the following lemma (the proof is in the supplementary material):

Lemma. If the rule to choose the division plane at each node is equivariant to similarity transformation σ, or more formally,

$$\textbf{choose-division-plane}(\sigma(P)) = \sigma(\textbf{choose-division-plane}(P)), \quad (3)$$

where **choose-division-plane**(P) is the procedure to choose the division plane on point set P, then the construction of the relaxed K-D tree is invariant to such a similarity transformation.

PointTree uses PCA to implement **choose-division-plane**(P), which is equivariant under any similarity transformation. As a result, our model is natively invariant to similarity transformations.

Affine (and Projective) Transformations. Interestingly, as empirically validated in Section 4, this invariance against similarity transformations enables PointTree also highly robust against affine and even more complicated projective transformations. To further improve the robustness to affine transformations, we introduce two additional strategies: pre-alignment and alignment network. Correspondingly, an input point cloud is fed forward the pre-alignment process and then the alignment network, before passing to our PointTree encoder (Figure 2).

Pre-alignment: A Normalization for Affine Transformations. We design a PCA-based pre-alignment scheme as "normalization" of affine transformed point clouds. For a centered point cloud $P \in \mathbb{R}^{n\times 3}$, applying PCA obtains $P = U\textbf{diag}(\Sigma)V^T$, where U is a 3×3 matrix, Σ is a length-3 vector which can be regarded as scalings of each axis, and V is an orthorgonal 3×3 matrix which can be regarded as a rotation. When we apply an affine transformation to a point cloud, the scaling and rotation can be arbitrary, and thus the normalization should not take these two pieces of information. So we disregard them in Σ and V, and take U as the normalized or pre-aligned point cloud (equivalent to normalizing P by applying another affine transformation $V\textbf{diag}(\Sigma^{-1})$). *Notably*, this pre-alignment scheme does not rely on the properties of K-D trees. As shown in Section 4, it is a general approach and can be used for a variety of existing point cloud encoders to improve their robustness. In our implementation, we found that applying pre-alignment *iteratively* further improves the performance, especially for part segmentation. Also, we have proven empirically that such a pre-alignment method is invariant to affine transformations. See the supplementary material for more details.

Alignment Network: A Learnable Component for Alignment. The pre-alignment is simply unlearnable. We also propose a *learnable* alignment network, inspired by PointNet [18]. This network takes a feature vector of a point cloud as input, feeds it into an MLP, and outputs a length-9 vector, which is reshaped to a 3×3 affine matrix to align the points. The encoder to generate the feature vector and the MLP are the learnable components of the alignment network.

The alignment network supports any encoder that outputs a feature vector, like a PointNet, another PointTree encoder, etc. Our default model uses the same architecture as "T-Net" in PointNet [18]. Note that such an alignment network is *not* designed for restoring the original point cloud (as a registration task). Instead, we only expect that it can learn to convert the input point cloud into an easier form for the following PointTree encoder. In Section 3.4, we also consider other variants of the alignment network that adopt different architectures.

Transformation Intensity Metric: Expected Angle Difference (EAD). To evaluate the intensity of a transformation, we propose a metric called expected angle difference (EAD). It is defined on two point clouds P and P' as follows: if we uniformly sample three different point indices $a, b, c \in [|P|]$, then the EAD is defined as the expected difference of $\angle P_b P_a P_c$ and $\angle P'_b P'_a P'_c$. Or,

$$\mathbf{EAD}(P, P') = \mathbb{E}_{a,b,c \in [|P|]} \left[\mathbf{angle\text{-}diff}(\angle P_b P_a P_c, \angle P'_b P'_a P'_c) \right]. \tag{4}$$

EAD is a metric for measuring the deformation of the transformed point cloud. By definition, similarity transformations hold $\mathbf{EAD}(P, P') = 0$, representing the minimum deformation – no deformation. Also, for the affine transformation which we randomly generated, the EAD is approximately $\frac{\pi}{8}$ (supplementary material). Meanwhile, the experiment shows that our pre-alignment scheme yields

$$\mathbf{EAD}(\mathbf{pre\text{-}align}(\mathbf{affine}_1(P)), \mathbf{pre\text{-}align}(\mathbf{affine}_2(P))) < 10^{-4}, \tag{5}$$

for any two affine transformations $\mathbf{affine}_1$ and $\mathbf{affine}_2$. This indicates that our pre-alignment is effective, which can normalize different affine transformations on a same point cloud to similar point clouds. More analysis about the EAD of transformations and pre-alignment methods are in the supplementary material.

3.3 Downstream Components

Classification: Point Cloud Features. In a classification task, each point cloud belongs to one class. Given a point cloud, the model should predict its class within all class candidates. For PointTree, the root node's information accounts for the whole point cloud, and we treat it as a global feature. We then build an MLP classifier that takes the root information $\text{info}(R)$ as input. The output will be the log-likelihood scores for C candidate classes (Figure 2).

General Segmentation: Point Features with Top-Down Information Flow. In a general segmentation (e.g., part or semantic segmentation) task, for a given point cloud, each point belongs to one of C_S candidate classes. The model should classify all points in the given point cloud. We design a segmentation decoder following KD-Net [10], as shown in Figure 4-b. The decoder is a K-D tree symmetric to the encoder. It follows a top-down flow, as opposite to the bottom-up flow in the encoder. Every node in the decoder has a feature called "carried information," representing the global-local relationship between inside and outside its subtree. Therefore, we can model the "role" of the subtree in the global shape. And when the node is leaf, it is exactly the "role" of the corresponding point in the global shape, which can be viewed as point feature.

Each node takes two inputs: the carried information from its ancestor, and the skip connection from the symmetric node in the encoder. The node merges these two inputs with one MLP, obtaining the carried information of itself. The top-down flow ends at leaf nodes and outputs the carried information of leaves. Such information is the feature of each point and is used for segmentation.

3.4 PointTree Variants

We introduce three variants of PointTree with different design of alignment networks and encoders. Note that, as mentioned in Section 3.2, the alignment network supports any encoder that outputs a point cloud feature. (1) **Default encoder ('Def')** uses T-Net in PointNet [18] as the alignment network. (2) **Encoder with K-D tree alignment ('KA')** introduces the default encoder as a stronger alignment network. (3) **ResNet-style encoder ('RNS')** is a ResNet-Style variant of PointTree with increased model capacity, by stacking more layers in a ResNet's style [8]. By connecting a default encoder and a general segmentation component, we can convert the $N \times 3$ input features to $N \times d$ intermediate features. We define such a connected structure as a "ResNet block," and stack a block followed by a default encoder to build a ResNet-style encoder. The output of each block is linked with the output of the previous block through a skip connection as in ResNet. For this variant, we can also treat the last encoder as the main encoder, and all previous encoders as part of the alignment network (since they mostly affect the input at the last layer of the last encoder). The detailed architectures are shown and explained in the supplementary material.

4 Experiments

Transformations. We evaluate our model on affine and projective transformed versions of existing datasets, including ModelNet40 [27], ShapeNetPart [3], and S3DIS [2]. For a dataset $D = \{P\}$ and a random distribution T of transformations, we construct the T-transformed dataset as follows: for each $P \in D$, we sample a fixed number ("augment time") of transformations $\{t\}$, and add all $t(P)$ in the dataset. Different point clouds will be applied to different transformations. We perform such process for D_{train}, D_{val}, and D_{test} separately with some specific "augment time," obtaining a full transformed dataset.

Table 1: PointTree significantly outperforms state-of-the-art point cloud encoders under affine transformations for object classification (instance-level overall accuracy (%)) on the affine transformed ModelNet40 dataset. With the proposed pre-alignment ('PA'), the performance of all methods consistently improves, and PointTree still achieves the best result. The results of baselines are obtained by running publicly released code on the transformed dataset. In addition, PointTree's accuracy has a lower standard deviation on different affine transformed datasets (supplementary material)

Type	Method	Affine w/ PA	Affine w/o PA
PointNet related	PointNet [18]	51.1	32.6
	PointNet++ [19]	72.7	47.8
K-D Tree-based	KD-Net [10]	65.3	23.1
	3DContextNet [33]	76.7	37.1
	PD-Net [31]	62.0	25.7
State-of-the-art	DGCNN [26]	79.4	57.4
	GBNet [20]	69.4	18.7
	GDANet [30]	72.8	15.6
	CurveNet [29]	82.1	59.3
	PointMLP [17]	82.3	63.1
	IT-Net [32] + DGCNN [26]	80.6	64.2
SO(3) Invariant/Equivariant	CloserLook [12]	82.4	64.9
	LGR-Net [36]	80.1	62.7
Ours	PointTree RNS	**84.1**	**73.1**

Baselines and PointTree Variants. For baseline models, we run the experiments with their official code by injecting the transformations into their data loaders. We keep their original optimal hyper-parameters, and train the models until convergence. We focus on four types of baselines: (i) PointNet related models (PointNet [18] and PointNet++ [19]), (ii) K-D tree-based models (KD-Net [10], 3DContextNet [33], and PD-Net [31]), (iii) recent state-of-the-art models (DGCNN [26], GBNet [20], GDANet [30], CurveNet [29], and PointMLP [17]), and (iv) SO(3) robust models (CloserLook [12] and LGR-Net [36]). As some baselines do not release their code on segmentation tasks, we only evaluate them on the classification task. We evaluate all three PointTree variants (Section 3.4).

Classification on ModelNet40. For the classification task, we evaluate our model on ModelNet40 [27]. ModelNet40 contains 9,843 point clouds for training and 2,468 point clouds for testing, and each of them belongs to one of 40 categories. We run the experiment on affine and projective transformations, and report the overall accuracy as our metric.

Affine Transformations. As shown in Table 1, our robust PointTree consistently outperforms all other point cloud models. Notably, PointTree significantly outperforms other models in the setting of affine without pre-alignment by more than 7%∼10%. This clearly shows that PointTree has much higher robustness against affine transformations. Our accuracy is also 8% and 36% higher than 3DContextNet [33], the previous best K-D tree-based model, in settings of affine with and without pre-alignment, respectively. This validates that the relaxed K-D tree is crucial to achieving the robustness that the original K-D tree is unable to. In addition, PD-Net [31] uses a similar PCA-based relaxed K-D tree as ours, but its design of feature aggregation highly relies on coordinates (by using the

Table 2: PointTree is robust even on the highly-challenging projective transformed ModelNet40 dataset, with and without pre-alignment ('PA'). It significantly outperforms all other models with a huge gap of 25% at instance-level accuracy (%)

Method	Projective w/ PA	Projective w/o PA
PointNet	15.4	4.3
DGCNN	47.3	6.2
PointMLP	49.9	4.1
CurveNet	37.6	5.6
PointTree RNS	**62.1**	**31.8**

Table 3: Ablation study results show that both pre-alignment ('PA') and alignment network improve our accuracy on ModelNet. Among the three variants, RNS achieves the best performance, but all of them outperform baselines in Table 1. Also, relaxed K-D tree is crucial for the transformation robustness in PointTree

Method			ModelNet40 Affine
PointTree RNS	w/ PA		84.1
PointTree KA	w/ PA		83.4
PointTree Def	w/ PA		82.7
PointTree RNS	w/o PA		73.1
PointTree KA	w/o PA		71.7
PointTree Def	w/o PA		70.2
PointTree Def	w/ PA	w/o Alignment Network	82.4
PointTree Def	w/ PA	w/ Original K-D Tree	68.8
PointTree RNS	w/ PA	w/ Concatenate-MLP	79.3

normal vector of children nodes' division plane), which wastes and nullifies the affine robustness of the tree structure, yielding a 13.6% worse accuracy than ours. Our method also consistently outperforms the SO(3) robust baselines, showing that the SO(3) robustness is not sufficient for coping with affine transformations.

Finally, by comparing the settings of affine with and without pre-alignment, we observe an at least 10% improvement in accuracy when applying pre-alignment in each baseline. This shows that pre-alignment is a general and effective approach to normalizing affine transformed point clouds.

Projective Transformations. Table 2 shows the accuracy of our model and top baselines on the projective transformed ModelNet40 dataset. In this experiment, PointTree significantly outperforms all baselines by more than 25%. For the most challenging setting, projective ModelNet40 without pre-alignment, PointTree still achieves a reasonable accuracy, while all other baselines can only obtain an accuracy that is a little higher than random guess.

Ablation Study. From the ablation study results in Table 3, we have the following observations. (1) By comparing the accuracy of the three PointTree variants in settings with and without pre-alignment, we observe that pre-alignment is helpful for our already-robust PointTree. (2) The three variants – Def, KA, and RNS – can be also interpreted as different types of the "alignment network" component in PointTree, from simplest to the most sophisticated. By comparing their accuracy, along with the variant "PointTree Def w/o Alignment Network," we can find that the improvement in alignment networks leads to higher ac-

Table 4: PointTree significantly outperforms baselines for part segmentation on affine transformed ShapeNetPart with pre-alignment, increasing class-level mIoU (%) by 7%

Method	airplane	bag	cap	car	chair	earphone	guitar	knife	lamp	laptop	motorbike	mug	pistol	rocket	skateboard	table	mIoU
PointNet [18]	69.7	53.5	57.8	59.2	84.1	44.9	62.9	41.7	61.9	64.5	31.8	71.0	50.3	35.4	46.1	76.6	56.9
PointNet++ [19]	66.5	73.7	58.6	37.6	72.0	71.5	85.9	74.4	72.5	51.0	29.1	74.9	54.1	41.6	57.1	71.9	62.1
DGCNN [26]	**89.0**	71.3	**91.2**	33.7	11.7	**94.3**	84.6	67.5	72.6	**92.9**	10.3	84.1	**83.7**	35.1	62.9	**97.3**	67.6
GDANet [30]	76.4	74.3	78.3	59.1	84.4	72.7	86.7	75.9	74.0	70.0	31.1	89.5	65.7	53.8	**75.6**	79.5	71.7
CurveNet [29]	75.8	64.7	79.3	60.7	86.9	59.5	86.2	72.4	74.8	69.3	24.6	89.5	63.5	39.2	60.1	78.4	67.8
PointTree RNS	83.6	**74.7**	82.5	**80.4**	**90.6**	66.5	**92.1**	**84.0**	**82.0**	88.3	**56.0**	**95.4**	78.0	**55.4**	68.9	81.2	**78.7**

Table 5: PointTree consistently outperforms state of the art for large-scale semantic segmentation on S3DIS, with notable margins of 2.2% and 6.7% on affine and projective transformed datasets, respectively, in overall accuracy (%) of Area5

Method	Affine w/ PA	Affine w/o PA	Projective w/ PA	Projective w/o PA
PointNet [18]	64.9	69.1	49.3	31.1
DGCNN [26]	80.0	70.8	67.1	57.8
PointTree Def	**82.2**	**74.9**	**73.8**	**61.2**

curacy. (3) RNS is the most powerful variant, due to its multiple layers and intermediate features. *Notably*, even the other variants have lower accuracy than RNS, they still outperform all the baselines in Table 1. (4) When we replace the relaxed K-D tree with the original K-D tree in PointTree, the performance experiences a dramatical drop by more than 10%. This indicates that the relaxed K-D tree is crucial for the robustness against transformations. (5) When we replace the implementation of **merge-info** from pointwise-maximum in Formula (2) to concatenate-MLP (concatenate o_l and o_r and apply an MLP), the accuracy clearly drops, showing that pointwise-maximum is a critical design choice.

Part Segmentation on ShapeNetPart. We test PointTree for the point cloud part segmentation task on ShapeNetPart [3]. It contains 16,881 point clouds in 16 classes. Each point belongs to one of 50 parts, where different classes have different sets of parts. We report the class-level mean intersection over union (mIoU) as the accuracy. Figure 5 visualizes two point clouds with PointTree and PointNet [18] as baseline. In both cases, the pre-alignment successfully normalizes the very flat affine transformed point cloud into a reasonable shape. For the lamp case, both PointNet and PointTree make a small mistake at the center of the lamp top (which is marked blue but should be green), while PointTree is more accurate at the bottom of the light pole. For the chair case, PointNet is performing badly, while our PointTree's accuracy is almost perfect.

Table 4 shows the segmentation results on affine transformed ShapeNetPart with pre-alignment. Our PointTree achieves a top mIoU over all baselines with an increase of more than 7%, and achieves best IoUs for more than half of the classes. This shows that PointTree is a general-purpose encoder that can work in both classification and segmentation tasks, being significantly more robust than other models. The results on projective transformed ShapeNetPart in the supplementary material demonstrate similar observations.

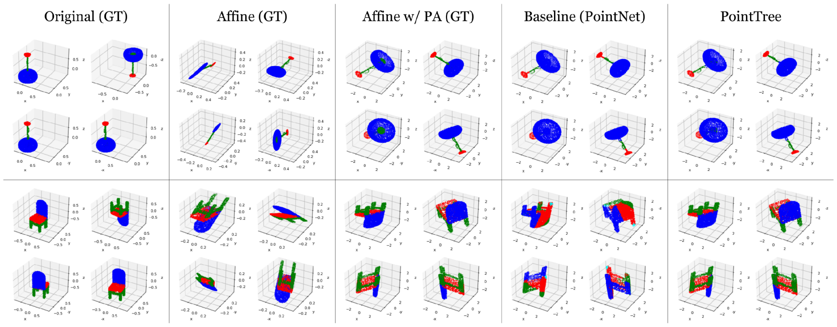


Fig. 5: Visualization of two point clouds in ShapeNetPart, showing that pre-alignment can successfully normalize the highly-deformed point clouds under affine transformations into reasonable shapes, and that PointTree works almost perfectly in this setting. Each grid of the table contains 4 point of views of a same point cloud. The third column shows the pre-aligned point clouds of the second column. The point clouds in first 3 columns are colored according to ground truth ('GT') segmentation, while last 2 columns are colored according to the segmentation outputs of PointNet and PointTree

Semantic Segmentation on S3DIS. Table 5 shows the semantic segmentation results on affine and projective transformed S3DIS [2] in settings with and without pre-alignment. Our PointTree achieves a top Area 5 overall accuracy over all baselines by large margins of at least 2.2%. PointTree is thus not only robust in simple single-object point cloud tasks like ModelNet40 and ShapeNetPart, but is also robust in complicated multi-object point cloud tasks like S3DIS.

5 Conclusion

In this paper, we proposed PointTree, a general-purpose point cloud encoder that is highly robust against affine and projective transformations. The key insight of PointTree is the use of relaxed K-D trees with PCA-induced similarity transformation-invariant construction. We further introduced pre-alignment, an effective and model-agnostic normalization scheme. Empirical evaluation shows that PointTree significantly outperforms state-of-the-art methods on various transformed datasets for classification and segmentation tasks. Notably, under affine transformations, the combination of PointTree with pre-alignment even achieves an accuracy that is close to the accuracy on the canonicalized point clouds. We hope our work could inspire more efforts on developing robust point cloud analysis models, and promote better exploitation of powerful K-D trees.

Acknowledgement. This work was supported in part by NSF Grant 2106825, the Jump ARCHES endowment through the Health Care Engineering Systems Center, the New Frontiers Initiative, the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign through the NCSA Fellows program, and the IBM-Illinois Discovery Accelerator Institute.

References

1. Agarwal, P.K., Arge, L., Danner, A.: From point cloud to grid DEM: A scalable approach. In: International Symposium on Spatial Data Handling (2006) 1
2. Armeni, I., Sax, A., Zamir, A.R., Savarese, S.: Joint 2D-3D-semantic data for indoor scene understanding. arXiv **abs/1702.01105** (2017) 3, 10, 14
3. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An information-rich 3D model repository. arXiv **abs/1512.03012** (2015) 2, 3, 10, 13
4. Duch, A., Estivill-Castro, V., Martinez, C.: Randomized K-dimensional binary search trees. In: International Symposium on Algorithms and Computation (1998) 3, 6
5. Gadelha, M., Wang, R., Maji, S.: Multiresolution tree networks for 3D point cloud processing. arXiv **abs/1807.03520** (2018) 4
6. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the KITTI vision benchmark suite. In: CVPR (2012) 1
7. Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L., Bennamoun: Deep learning for 3D point clouds: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **43**, 4338–4364 (2021) 4
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016) 10
9. Klein, G., Murray, D.: Parallel tracking and mapping for small ar workspaces. In: IEEE and ACM International Symposium on Mixed and Augmented Reality (2007) 1
10. Klokov, R., Lempitsky, V.: Escape from cells: Deep Kd-Networks for the recognition of 3D point cloud models. In: ICCV (2017) 3, 4, 5, 10, 11
11. Lei, H., Akhtar, N., Mian, A.S.: Octree guided CNN with spherical kernels for 3D point clouds. In: CVPR (2019) 4
12. Li, F., Fujiwara, K., Okura, F., Matsushita, Y.: A closer look at rotation-invariant deep point cloud analysis. In: ICCV (2021) 4, 11
13. Li, X., Li, R., Chen, G., Fu, C.W., Cohen-Or, D., Heng, P.A.: A rotation-invariant framework for deep point cloud analysis. IEEE Transactions on Visualization and Computer Graphics pp. 1–1 (2021) 4
14. Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B.: PointCNN: Convolution on X-transformed points. In: NeurIPS (2018) 4
15. Liu, Y., Fan, B., Meng, G., Lu, J., Xiang, S., Pan, C.: DensePoint: Learning densely contextual representation for efficient point cloud processing. arXiv **abs/1909.03669** (2019) 4
16. Lv, X., Wang, B., Dou, Z., Ye, D., Wang, S.: LCCNet: LiDAR and camera self-calibration using cost volume network. In: CVPRW (2021) 2
17. Ma, X., Qin, C., You, H., Ran, H., Fu, Y.: Rethinking network design and local geometry in point cloud: A simple residual MLP framework. arXiv **abs/2202.07123** (2022) 1, 2, 4, 11
18. Qi, C.R., Su, H., Kaichun, M., Guibas, L.J.: PointNet: Deep learning on point sets for 3D classification and segmentation. In: CVPR (2017) 1, 2, 3, 4, 7, 9, 10, 11, 13
19. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: Deep hierarchical feature learning on point sets in a metric space. In: NeurIPS (2017) 3, 4, 7, 11, 13
20. Qiu, S., Anwar, S., Barnes, N.: Geometric feedback network for point cloud classification. arXiv **abs/1911.12885** (2019) 11

21. Que, Z., Lu, G., Xu, D.: VoxelContext-Net: An octree based framework for point cloud compression. arXiv **abs/2105.02158** (2021) 4
22. Riegler, G., Ulusoy, A., Geiger, A.: Octnet: Learning deep 3D representations at high resolutions. In: CVPR (2017) 4
23. Shen, W., Zhang, B., Huang, S., Wei, Z., Zhang, Q.: 3D-rotation-equivariant quaternion neural networks. In: ECCV (2020) 4
24. Siekański, P., Paśko, S., Malowany, K., Malesa, M.: Online correction of the mutual miscalibration of multimodal VIS–IR sensors and 3D data on a UAV platform for surveillance applications. Remote Sensing **11**(21) (2019) 2
25. Sun, J., Zhang, Q., Kailkhura, B., Yu, Z., Xiao, C., Mao, Z.M.: Benchmarking robustness of 3D point cloud recognition against common corruptions. arXiv **abs/2201.12296** (2022) 4
26. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph CNN for learning on point clouds. ACM Transactions on Graphics **38**(5) (2019) 4, 11, 13
27. Wu, Z., Song, S., Khosla, A., Tang, X., Xiao, J.: 3D shapenets for 2.5D object recognition and next-best-view prediction. arXiv **abs/1406.5670** (2014) 2, 3, 10, 11
28. Xiang, B., Tu, J., Yao, J., Li, L.: A novel octree-based 3-D fully convolutional neural network for point cloud classification in road environment. IEEE Transactions on Geoscience and Remote Sensing **57**, 7799–7818 (2019) 4
29. Xiang, T., Zhang, C., Song, Y., Yu, J., Cai, W.: Walk in the cloud: Learning curves for point clouds shape analysis. In: ICCV (2021) 11, 13
30. Xu, M., Zhang, J., Zhou, Z., Xu, M., Qi, X., Qiao, Y.: Learning geometry-disentangled representation for complementary understanding of 3D object point cloud. In: AAAI (2021) 11, 13
31. Yi, L., Shao, L., Savva, M., Huang, H., Zhou, Y., Wang, Q., Graham, B., Engelcke, M., Klokov, R., Lempitsky, V.S., Gan, Y., Wang, P., Liu, K., Yu, F., Shui, P., Hu, B., Zhang, Y., Li, Y., Bu, R., Sun, M., Wu, W., Jeong, M., Choi, J., Kim, C., Geetchandra, A., Murthy, N., Ramu, B., Manda, B., Ramanathan, M., Kumar, G., Preetham, P., Srivastava, S., Bhugra, S., Lall, B., Häne, C., Tulsiani, S., Malik, J., Lafer, J., Jones, R., Li, S., Lu, J., Jin, S., Yu, J., Huang, Q., Kalogerakis, E., Savarese, S., Hanrahan, P., Funkhouser, T.A., Su, H., Guibas, L.J.: Large-scale 3D shape reconstruction and segmentation from ShapeNet core55. arXiv **abs/1710.06104** (2017) 4, 11
32. Yuan, W., Held, D., Mertz, C., Hebert, M.: Iterative transformer network for 3D point cloud. arXiv **abs/1811.11209** (2018) 4, 11
33. Zeng, W., Gevers, T.: 3Dcontextnet: K-d tree guided hierarchical learning of point clouds using local contextual cues. arXiv **abs/1711.11379** (2017) 3, 4, 5, 11
34. Zhang, X., Zhu, S., Guo, S., Li, J., Liu, H.: Line-based automatic extrinsic calibration of lidar and camera. In: ICRA (2021) 2
35. Zhang, Y., Rabbat, M.G.: A graph-CNN for 3D point cloud classification. In: ICASSP (2018) 4
36. Zhao, C., Yang, J., Xiong, X., Zhu, A., Cao, Z., Li, X.: Rotation invariant point cloud classification: Where local geometry meets global topology. arXiv **abs/1911.00195** (2019) 4, 11