ABSTRACT

In this study, we present an analysis of model-based ensemble learning for 3D point-cloud object classification and detection. An ensemble of multiple model instances is known to outperform a single model instance, but there is little study of the topic of ensemble learning for 3D point clouds. First, an ensemble of multiple model instances trained on the same part of the ModelNet40 dataset was tested for seven deep learning, point cloud-based classification algorithms: PointNet, PointNet++, SO-Net, KCNet, DeepSets, DGCNN, and PointCNN. Second, the ensemble of different architectures was tested. Results of our experiments show that the tested ensemble learning methods improve over state-of-the-art on the ModelNet40 dataset, from 92.65% to 93.64% for the ensemble of single architecture instances, 94.03% for two different architectures, and 94.15% for five different architectures. We show that the ensemble of two models with different architectures can be as effective as the ensemble of 10 models with the same architecture. Third, a study on classic bagging (i.e. with different subsets used for training multiple model instances) was tested and sources of ensemble accuracy growth were investigated for best-performing architecture, i.e. SO-Net. We also investigate the ensemble learning of Frustum PointNet approach in the task of 3D object detection, increasing the average precision of 3D box detection on the KITTI dataset from 63.1% to 66.5% using only three model instances. We measure the inference time of all 3D classification architectures on a Nvidia Jetson TX2, a common embedded computer for mobile robots, to allude to the use of these models in real-life applications.

Keywords Point Cloud · Point Set · Classification · Detection · Ensemble Learning · 3D Deep Learning

1 Introduction

Over the last few years with the rapid development of sensor technology, processing of three–dimensional data (3D) has become an important topic of research. High quality, long range laser scanners are widely used in autonomous cars [1], and the availability of cheap RGB-D sensors has resulted in significant progress in 3D mobile robots perception [2]. Accurate object detection, segmentation, and classification from 3D point-clouds are challenging problems, especially so in real-world settings, and crucial for performing robotic tasks.

A lot of the handcrafted approaches to 3D point cloud analysis have been developed previously [3, 4, 5]; however, in recent years, deep learning – based approaches have increased in popularity [6, 7, 8, 9]. Results of deep learning models are strongly correlated to the available amount of data and to the quality of used regularization techniques [10].

In point cloud classification task we assume that the object is already segmented, which means that all the points belong to that single-class object. There are 3 main approaches to point cloud classification:

- 3D ConvNets — point cloud is converted to voxel grid with a given resolution. This approach is not memory efficient in case of large volumes. The sparsity of 3D data leads to inefficient and redundant computation.
However, some octree- or kd-tree–based approaches reduce these disadvantages and provide encouraging results [6, 11].

- Rendering a set of 2D views of the 3D object — the problem is transformed into a set of 2D vision problems. View-pooling layer [12] may be used to aggregate features from different views. This technique leverages the performance of 3D ConvNets, but the loss of information during rendering makes this approach impractical in point-level segmentation task [7, 13].

- Direct point cloud processing — architectures that directly process point-clouds in an order-invariant manner, first presented by the PointNet architecture. It can be adapted to the analysis of different kinds of problems ranging from an estimation of population statistics [14], anomaly detection in piezometer data of embankment dams [15], to cosmology [16].

In this paper, we focus on the direct point cloud processing, because such architectures can perform well not only in classification, but also in segmentation and detection tasks. Seven architectures are used in our experiments: PointNet [8], PointNet++ [17], SO-Net [18], KCNet [19], DeepSets [20], DGCNN [21], and PointCNN [22]. We chose these because of their prominence and the availability of author’s implementations that are open to the research community.

For object classification, two types of datasets can be considered. The first type is based on 3D CAD models: PrincetonSB [23], ModelNet [11], ShapeNet [24], and many others. The second type are datasets of 3D objects/scenes acquired from the real world with depth sensors [1, 25, 26, 1]. In this work, we focus on ModelNet40 [11], because it is one of the most popular benchmarks for object classification. It contains 40 classes of objects’ CAD models. For real-world applications, the KITTI dataset [1] is the most prominent and widely used benchmark for 3D perception of autonomous vehicles and thus is our focus in this study.

Ensemble learning [27] increases performance of the prediction with the help of multiple models. Several methods are reported in the literature: bagging, boosting, stacking, a bucket of models, Bayesian methods, and many others. In this article, we focus on bagging, also known as bootstrap aggregating. We test three voting methods: direct output averaging, soft voting, and hard voting. We compare the ensemble of model instances trained on the same training set and its different subsets and evaluate their performance.

Knowledge distillation [28] is a training procedure which learns a so-called student model (usually fast mobile network) to mimic a teacher (usually ensemble of complex models) on a large unlabelled dataset rather than ground truth labels. An ensemble is often useful as a teacher model as its execution time is not crucial [29]. Experiments with knowledge distillation are beyond the scope of this work, which is focused on leveraging the ensemble results.

Our experiments show that an ensemble of neural networks trained on the whole training set is better than bagging using random parts of the training set. An ensemble of different model with different architectures can even further improve the prediction performance.

We also count the number of trainable parameters and inference execution times on NVIDIA Jetson TX2 platform for each approach. According to previous studies [30, 31], using the Jetson platform as a high-level driver is a reasonable choice for energy-efficient mobile robotic applications.

2 Related Work

To the best of our knowledge, there are no studies reporting on the strict influence of different ensemble methods and the number of aggregated models to the prediction accuracy for direct point cloud classification architectures. Su et al. [32] studied a model combining different types of representations, but since there are fast advances in this field, there are more and more models in the point cloud classification zoo. A previous article [33] has reported significant performance gain while using an ensemble of 10 instances of one voxel-based, deep learning architecture, introduced in that article.

This section starts with a brief description of each tested architecture. We compare the reported results with test accuracy reproduced in our experiments. The whole setup with exact versions of all libraries and code version used has been shared online. The difference with original reports may be caused by some implementation details, hyper–parameters, or test methodology (reporting maximum vs. average score). Finally, we briefly describe bagging.

2.1 DeepSets

DeepSets [20] is one of the two independently developed (along with PointNet) first deep learning approaches to shape classification using bare point sets. The general idea is quite similar in both methods, but DeepSets focuses primarily on

[https://github.com/dkoguciuk/ensemble_learning_for_point_clouds](https://github.com/dkoguciuk/ensemble_learning_for_point_clouds)
producing the permutation equivariance layer, which consists of three operations: \(\phi\), sum, and \(\rho\). According to the paper, \(\phi\) could be an arbitrary neural network architecture applied iteratively over every point in the point cloud, and the output should be summed along the set dimension. The reduced vector can then be passed into the multilayer perceptron (MLP) working as a classifier.

Given the symmetry in weight sharing, the final features of the whole point set (after the summation) are invariant to the ordering of the input. This article demonstrates a strong mathematical background, many sample applications, and great results on the ModelNet40 dataset. Despite this, there is no direct successor of the model in the literature.

### 2.2 PointNet

The basic idea of PointNet [8] is to learn a spatial encoding of each point using a series of nonlinear mappings and then aggregate all point features to one global point cloud signature. The first part plays a similar role to the \(\phi\) operation in DeepSets (there are some slight differences in the weight sharing scheme), and the second one (symmetric function) is also DeepSets alike, but the authors suggest using max pooling operation as the one achieving the best performance.

The model is also invariant to the order in which the points are presented, which can directly operate on point clouds without any additional preprocessing such as spatial partitioning or graph construction. Moreover, the model is extremely robust to deformation and noise, but by its design, it suffers from not being able to detect small local structure details; thus, it leaves big space for modifications.

### 2.3 PointNet++

PointNet++ is an extended version of original PointNet architecture [17], where authors made hierarchical feature extraction by building a pyramid-like aggregation scheme to combine features from multiple scales. There are three steps on each pyramid level: sampling, grouping, and feature extraction. The first two steps consist of partitioning the input point cloud into overlapping local regions by the distance metric of the underlying space. The third step is learning a higher dimensional representation of an input region with the so-called local learner which is, naturally, a standard PointNet model. There are three such pyramid levels in the original PointNet++ article, which produce the features of the whole point set.

### 2.4 SO-Net

SO-Net [18] is another hierarchical feature extraction model based on PointNet approach, but it has a different sampling and grouping strategy in comparison to PointNet++. The main idea is to build a Self-Organizing Map (SOM) to model the spatial distribution of the input point cloud and then assign each point into \(k\) nearest SOM nodes, which play a similar role to sampling and grouping steps in PointNet++. Each local region is processed with a local pointnet-like learner, and a channel-wise max pooling operation is applied to aggregate point features to node signatures. Now, each SOM node with its features is processed with the second level learner and again aggregated with max-pool into a feature vector that represents the whole input point cloud.

### 2.5 KCNet

The main idea of the KCNet model [19] is to construct a kernel correlation layer as an affinity measure between a query point with its neighbors and kernel points, where the latter are allowed to move and adjust during training freely. KCNet uses several kernels at the local level to augment the original 3D coordinates’ input of the PointNet architecture. The second modification to the plain PointNet model is a recursive max-pooling operation performed in the neighborhood of each node.

### 2.6 DGCNN

Instead of generating the point embedding directly from the point coordinates, the DGCNN [21] introduces EdgeConv operation, which incorporates a point’s neighboring structure. For each point of a point cloud, they construct a local neighborhood graph and apply deep learning feature extraction on edges of this graph. EdgeConv is designed to be invariant to the ordering of neighbors, and thus is permutation invariant.

The operation could be applied hierarchically just like in the traditional convolutional networks for 2D images, but the authors propose to build the graph of neighboring points for each layer independently.
2.7 PointCNN

The PointCNN [22] is using MLP on the local neighborhood to organize points into latent canonical order. Learning, so-called, $\mathcal{X}$-transformation results not only in permuting input but also in weighing the features associated with the points. Next, an element-wise product and sum operations of the typical convolution operation are subsequently applied on the $\mathcal{X}$-transformed features.

The above-described operations could be applied hierarchically: after each convolution, a subset of points are retained by downsampling, thus contain richer information aggregated the expanding neighborhood.

2.8 Comparison of Models

The general idea behind PointNet and DeepSets approaches is similar, but they differ mostly in weights sharing scheme in the MLP network. Both are pioneers in the field but do not explicitly use local structure information.

PointNet++ and SO-Net both hierarchically apply PointNet but differ in the sampling and grouping strategy. KCNet adds local structure information, which is learnable — not designed by hand like in PointNet++ or SO-Net. However, the kernel size is fixed here, which could not reveal the spatial distribution like SO-Net.

Last two models incorporate information of the local neighborhood of a query point by building a graph structure around it and by applying some kind of graph convolution. DGCNN learns feature extraction between a pair of neighboring points, where PointCNN tries to organize the nearby points in the canonical order.

2.9 Frustum PointNet

The main author of PointNet and PointNet++ extended his network into a 3D object detection framework called Frustum PointNet [24]. The main idea is to combine both mature 2D and novel 3D approaches by splitting the task into three main stages: 3D frustum proposal based on 2D object detection, 3D instance segmentation, and 3D bounding box estimation. The modules are based on PointNet (denoted as v1) or PointNet++ (v2) architectures.

In the first stage, 2D CNN object detector is used to generate 2D region proposals. Then each region is lifted up to 3D and thus become frustum proposal, containing point cloud $I$ — all points in the LiDAR point cloud which lie inside the 2D region when projected onto the image plane. The output point cloud $I$ is then fed into 3D instance segmentation network with binary output $P = f_P(\theta_p, I, c)$ (where $\theta_p$ are the model instance weights, $c$ is class of the object and $f_p$ is a model function) meaning if the given point is a part of a 3D object or not, which assumes there is only one meaningful object in the frustum. Points belonging to the object form a point cloud $O = f_O(I, P)$. Then, the T-Net module find transformation $T = f_T(\theta_T, O)$ centering $O$. Centered point cloud is denoted as $C = f_C(O, T)$. In the last step, $C$ is used to estimate amodal 3D bounding box of the object $B = f_B(\theta_B, C)$, which is finally transformed to the global frame. Amodal bounding box $B$ is described by its position, size, and heading angle. Size and heading are represented as discrete probability distributions, not raw values. Summing up, the network prediction can be denoted as:

$$
B = f_B(\theta_B, f_C(O, f_T(\theta_T, O)))
$$

$$
O = f_O(I, f_P(\theta_P, I, c))
$$

(1)

2.10 Ensemble Learning with Bagging and Boosting

An ensemble consists of a set of individually trained models, whose predictions are combined. It is well known that ensemble methods can be used for improving prediction performance [27, 55].

Individually trained models may be trained using different training sets. In bagging, the training set is selected independently for each classifier from the full training set. The selected set can be a subset of the entire training set (later referred to as bagging without replacement) or can have the same size, but some samples repeated several times (bagging with replacement). However, the result of neural network training depends on several random factors so that the ensemble can consist of classifiers trained on the same training set, which we refer to as a simple ensemble.

Boosting trains a series of classifiers, with the training set (or samples loss weights) of the next classifier focusing on the samples with a higher error for the previous classifier. This can reduce errors, but noise in the training data often results in boosting overfitting [27].

Given the output of the individual classifier, the output of the ensemble can be calculated in different ways. Boosting uses individual weight for each of the classifier in series. Stacking trains a learning algorithm to combine predictions. In bagging and simple ensembles, all classifiers are equivalent, so three aggregation methods are commonly used in
classification task: direct output averaging, soft voting (sum of activation of all hypothesis for each sample equals to one) and hard voting (each classifier output is in one-hot form, i.e. each classifier votes for one hypothesis).

3 Ensemble Learning for 3D Object Recognition

We performed several experiments during the conduct of this study. Seven deep network architectures were selected: PointNet, PointNet++, SO-Net, KCNet, DeepSets, DGCNN, and PointCNN. All networks were tested on one task, i.e. ModelNet40 object classification. All these networks take a raw 3D point cloud as an input and output a vector of class scores for a given object, which can be denoted as follows:

\[ F: \{p_i \in \mathbb{R}^3, i = 1, \ldots, N\} \rightarrow \mathbb{R}^C \]  

where \( N \) is the number of points in the point cloud, and \( C \) is the number of classes in the classification task.

One set of hyper–parameters is selected for each network based on the authors’ settings. 10 model instances are trained for each architecture. The influence of the number of models in the simple ensemble is tested for each architecture, which is described in subsection 3.1

Different architectures achieve the best result for different object classes. This suggests studying if the ensemble of different models can outperform every single model. Such a comparison for PointNet, PointNet++, KCNet, DGCNN, PointCNN, and DeepSets was not included due to the challenges in technical implementation.

An SO-Net architecture achieved the highest accuracy during our experiments, so it was selected for further bagging tests. We tested the classification accuracy for bagging with and without replacement in subsection 3.3.

The ensemble of several model instances is computationally expensive. However, a deep network architecture can be viewed as an encoder (transforming sample to a feature vector) followed by a classifier (e.g. MLP, calculating class probabilities based on a feature vector). The question arises, whether classification accuracy can be improved by an ensemble of classifiers, based on the same feature vector. This was tested for SO-Net architecture according to subsection 3.4.

Our work shows that a simple ensemble of several model instances of the same architecture increases classification performance. Random factors cause differences between model instances. Influence of each factor is evaluated in subsection 3.5.

An ensemble of three model instances of Frustum PointNet was tested on the KITTI dataset. This is described in section 3.6. Note, that Frustum PointNet is a pioneering approach, in the sense that it performs detection + classification on raw LiDAR scans, so it cannot be compared directly with the other models presented.

3.1 Simple Ensemble of Model Instances

We experiment with the simple ensemble, which is a special case of bagging, with the full training dataset being used to train every model instance. The ensembling is performed by averaging the raw output activation, soft voting, or hard voting (denoted as \( F, S, \) and \( H \) respectively):

\[ F_e = \{ f_i = \text{mean}(f_{kj}, k = 1, \ldots, K), j = 1, \ldots, C \} \]  

\[ S_e = \{ f_i = \text{softmax}(f_k)_j, k = 1, \ldots, K), j = 1, \ldots, C \} \]  

\[ H_e = \{ f_i = \text{mean}(\text{onehot}(\text{argmax}(f_k)))_j, k = 1, \ldots, K), j = 1, \ldots, C \} \]

Where each model instance is denoted as:

\[ F_k = \{ f_{kj}, j = 1, \ldots, C \} \]

where \( k \) is a model index. For each architecture, 10 model instances were trained. Tests were performed for \( K = 1, \ldots, 10 \). For each value of \( K \), all \( \binom{10}{K} \) model instances’ combinations were selected. Mean and standard deviation for each value of \( K \) are reported in the experimental results.

Figure 1 shows the comparison of voting methods. Raw activation averaging equals approximately to soft-voting and usually outperforms hard-voting. For simplicity, only activation averaging is used in the rest of this article.

Figure 2 presents the instance and mean class accuracy as a function of used models for each architecture, along with their standard deviation. As one can observe, with the increasing number of models in the ensemble, the classification
accuracy is slightly rising, and the standard deviation of classification accuracy is getting smaller, which means that the output is more stable and not so much dependent on a single learning session.

Table 1 shows the numerical comparison of classification accuracy increase between all approaches. The simple ensemble of KCNet instances has a noticeably higher increase in the classification accuracy (2.52%), then second SO-Net (0.99%) and other architectures (with mean instance accuracy increase equal to 0.50%).

Figures 3 and 4 show the loss or gain in the classification accuracy per every class between plain version and simple ensemble for all tested approaches. Some classes are easy to classify and all approaches achieve 100% accuracy, for example, airplane and laptop. However, interestingly, there are classes where the PointNet approach does better than others, despite the smallest accuracy of overall classification, for example, glass box and stool. This suggests that different methods can be focused on various aspects of point clouds, in particular, focusing on local structure leads to overfitting for some classes with more discriminative global shape.

One can ask one more interesting question about those approaches and their ensembles: in how many classes a particular model has the highest accuracy? Figure 5 answers that question and shows how this number is changed after simple ensemble.

Figure 1: Dependency between instance classification accuracy (left) or mean class accuracy (right) and $k$-number of ensemble models for a given voting method. For visibility, only results for best-performing architecture and mean result of all architectures are plotted. Results for soft-voting and activation ensemble are approximately equal, usually outperforming hard-voting.

Figure 2: Dependency between instance classification accuracy (left) or mean class accuracy (right) and $k$-number of ensemble models. We have learned each approach independently 10 times (70 different models), then for each $k$ possible number included in the ensemble classifier we have randomly chosen ten different $k$-subsets and have reported mean accuracy with its standard deviation across those $k$-subsets. As one can observe, using the ensemble learning makes the output more stable and classification accuracy rise slightly.
Figure 3: Plain class classification accuracy, its gain (green), or loss (red) for all architectures for less accurate classes. Please note the flower pot class is the hardest class to classify, and almost all architectures using point neighborhood are doing slightly worse with ensemble learning here. This could suggest that the general shape for flower pot is more meaningful than local structure information.

ensemble (if \( N \) classes reach same best accuracy, each of them scores \( 1/N \) in this rank). Note that the order of architectures is different than that given in Table I.

3.2 Ensemble of Different Models

We evaluate ensembles of pairs of different models. First, an ensemble of output scores were calculated for pairs of architectures according to the following formula:

\[
F_{pair} = k_1 \cdot F_1 + k_2 \cdot F_2
\]

\[
k_1 + k_2 = 1
\]

\[
k_1 = 0.1, 0.2, \ldots, 0.9
\]

Note that each output of the architecture is scaled to have identity standard deviation for the training set. The ensemble of different models improved both instance and mean class accuracy. Top pairs consist of SO-Net model with higher weight and the second architecture. Ensemble results are calculated for 10 model instances: 5 of one architecture and 5
Figure 4: Plain class classification accuracy, its gain (green) or lose (red) for all architectures for more accurate classes (with accuracy between 80 and 100%). Please note that many classes, for example, *bed* or *cone*, all architectures achieves better classification accuracy using ensemble learning.
Figure 5: Number of classes where certain architecture is the best for a plain version, its gain (green), or loss (red). High gain of GDCNN and PointCNN architectures suggests they can be much better in classifying some classes and much worse in other ones.

of the other. Table 2 shows the most interesting results. As two model instances are used in the ensemble to calculate its plain accuracy, results for two SO-Net instances ensemble are plotted for reference.

Ensemble of all considered architectures with the best-performing SO-Net was tested according to the following formula:

\[
F_{all} = k_{pointnet} \cdot F_{pointnet} + k_{pointnet+} \cdot F_{pointnet+} + k_{kcn} \cdot F_{kcn} + \\
+ k_{dgcnn} \cdot F_{dgcnn} + k_{pointcnn} \cdot F_{pointcnn} + k_{so-net} \cdot F_{so-net} \\
+ k_{pointnet} + k_{pointnet+} + k_{kcn} + k_{dgcnn} + k_{pointcnn} + k_{so-net} = 1
\]

\(k_{pointnet} = 0.0, 0.05, \ldots, 0.35\)

\(k_{pointnet+} = 0.0, 0.05, \ldots, 0.35\)

\(k_{kcn} = 0.0, 0.05, \ldots, 0.35\)

\(k_{dgcnn} = 0.0, 0.05, \ldots, 0.35\)

\(k_{pointcnn} = 0.0, 0.05, \ldots, 0.35\)

\(k_{so-net} > 0.4\)

Note that this is the ensemble of different architectures including only one training instance of each architecture. The obtained models are further tested in the aspect of multiple training instances learning as described in subsection 3.1.

The ensemble of all architectures with the major role of SO-Net, achieves the highest overall accuracy. The results of the ensemble are calculated for five instances for each architecture with the nonzero factor. Table 3 shows the most interesting results.

### 3.3 Ensemble Learning with Model Bagging

For the SO-Net architecture, which achieves the highest overall accuracy, model bagging was tested. For bagging with replacement, 10 training sets were generated by randomly sampling with replacement of a number of samples equal to...
Table 3: Example results for ensemble of models with different architecture.

| k_{pointnet} | k_{pointnet++} | k_{kcnet} | k_{dgcnn} | k_{so-net} | Instance accuracy mean | Instance accuracy ensemble | Class accuracy mean | Class accuracy ensemble |
|-------------|----------------|------------|-----------|-------------|------------------------|--------------------------|------------------------|------------------------|
| -           | 0.1            | -          | -         | 0.2         | 93.73%                 | 94.13%                   | 91.24%                 | 91.77%                 |
| 0.05        | 0.05           | -          | -         | 0.25        | 93.74%                 | 94.14%                   | 91.17%                 | 91.57%                 |
| -           | 0.05           | -          | 0.05      | 0.3         | 93.88%                 | 94.14%                   | 91.52%                 | 91.92%                 |
| -           | 0.05           | -          | -         | 0.2         | 93.65%                 | 94.15%                   | 91.14%                 | 91.76%                 |
| 0.05        | 0.05           | 0.3        | -         | 0.7         | 93.67%                 | 94.15%                   | 91.12%                 | 91.71%                 |
| -           | 0.3            | -          | 0.05      | -           | 93.73%                 | 94.04%                   | 91.66%                 | 92.20%                 |
| 0.05        | 0.3            | -          | 0.05      | -           | 93.78%                 | 94.04%                   | 91.79%                 | 92.20%                 |
| -           | 0.15           | -          | -         | 0.05        | 93.76%                 | 94.03%                   | 91.69%                 | 92.21%                 |
| -           | 0.15           | -          | 0.1       | -           | 93.76%                 | 94.05%                   | 91.64%                 | 92.22%                 |
| -           | 0.2            | -          | 0.1       | -           | 93.68%                 | 94.06%                   | 91.60%                 | 92.24%                 |

the size of the training set. For bagging without replacement, $S = 9$ training subset sizes were used in experiments, denoted as follows:

$$s_i = k \cdot \text{size}(T_{train}), k = 0.1, 0.2, \ldots, 0.9$$  (11)

For each $s_i$, 10 training set splits were generated (sampled without replacement), and one model instance was trained. Output of each model for a given $s_i$ was aggregated as detailed in subsection 3.1.

Figure 6: Results of the simple ensemble, bagging without replacement for ten different training subset sizes and bagging with replacement. Results for SO-Net architecture, instance (left) and class (right) classification accuracy. Error bars not plotted to prevent jamming. The biggest gain is achieved for the smallest training subset size, but the overall classification accuracy is the best for simple ensemble (aka bagging with replacement).

Figure 6 shows the results. The biggest gain is achieved for the smallest training subset size. Accuracy increase for bagging with replacement is higher than that of without replacement. However, none of the bagging methods outperforms simple ensemble in the given task.

3.4 Simple Ensemble of Last Layers

The SO-Net architecture consists of the explicitly defined encoder (computationally expensive) and (fast) classifier. Now, we can check whether the accuracy growth of ensemble learning or SO-Net architecture is determined mostly by encoder or classifier part. In the case of the latter, one could learn the ensemble of classifiers only and thus save learning time by a significant factor. To check this, for each of the 10 SO-Net encoder instances, 5 additional classifiers were trained, with the same hyper–parameters, constant encoder weights, and 31 training epochs. The average result of 5–classifier ensemble is compared to the average result of model with a single classifier (both averages are calculated for 10 encoder instances).

Table 4 shows the results. According to the results, the encoder causes the major advantage of the ensemble. This means that computationally cheaper classifier ensemble does not result in rewarding accuracy gain.
Table 4: The influence of classifiers ensemble with one encoder for SO-Net architecture. Five classifiers were trained for each of the 10 encoders. As one can see, the computationally cheaper classifier ensemble does not result in rewarding accuracy gain.

| Parameter                  | Mean     | Standard Deviation |
|----------------------------|----------|--------------------|
| Instance accuracy mean     | 92.43%   | 0.19%              |
| Instance accuracy ensemble | 92.69%   | 0.28%              |
| Class accuracy mean        | 89.80%   | 0.21%              |
| Class accuracy ensemble    | 89.98%   | 0.28%              |
| Instance accuracy increase | 0.25%    | 0.12%              |
| Class accuracy increase    | 0.18%    | 0.12%              |

3.5 Influence of Random Factors in Simple Ensembles

We identified four random factors in SO-Net model training:

- The order of training samples and random data augmentation (note that eliminating this factor means that samples are shuffled between training epochs but in the same way for every model instance);
- Initial values of weights and biases of the neural network;
- Random dropout regularization (eliminating this factor means fixing the dropout seed so certain neuron would be dropped, for example, always in epoch number 3, 7, 17, etc.);
- Random order of massively parallel computations, resulting in different summation order, which is not alternating for floating point numbers.

The first three factors can be eliminated, whereas the last one cannot be eliminated. To verify the influence of each factor, five SO-Net model instances were trained for each configuration with one, two, or three random factors eliminated.

The experiments were time-consuming (35 additional training sessions), but the results, presented in Table 5, are coarse because only one constant order of values for each factor was considered. However, one can observe that the increase in the accuracy can be observed even if all model instances in the ensemble were trained with the same training data order and augmentation, initial weights, and dropout order. This leads to the conclusion that for SO-Net architecture, diversity in models is caused mainly just by the numerical issues of massively parallel computations.

3.6 Ensemble Methodology for Frustum PointNet

Since the first step of Frustum PointNet approach use 2D CNN object detector, which is beyond the scope of this paper, we look at ensemble learning for the three point cloud processing modules of the network. A single experiment on the KITTI dataset was performed to verify conclusions based on ModelNet40 and to plan further work on knowledge distillation with ensemble learning for real-world 3D data.

Frustum PointNet consists of three trainable modules. Ensembling can be used either for the last bounding-box predicting module or for all modules (denoted as $B_L$ and $B_E$, respectively). Results for other configurations (e.g. ensemble of only segmentation modules) have been shared online. Given $N$ independently trained model instances, $B_L$

Table 5: The influence of random factors in model instances training to ensemble accuracy gain. One can observe that accuracy increases even if results of massively parallel computations are the only one (irremovable) random factor. One can observe all removable random factors in the training procedure have little influence on classification accuracy in ensemble learning.

| Training data | Weights initialization | Dropout | Instance accuracy mean | Instance accuracy ensemble | Class accuracy mean | Class accuracy ensemble | Instance accuracy increase | Class accuracy increase |
|---------------|------------------------|---------|------------------------|----------------------------|--------------------|------------------------|--------------------------|------------------------|
| const         | const                  | const   | 92.34%                 | 93.19%                     | 89.58%             | 90.96%                 | 0.85%                    | 1.38%                  |
| const         | const                  | random  | 92.48%                 | 93.11%                     | 89.89%             | 90.36%                 | 0.63%                    | 0.48%                  |
| const         | random                 | const   | 92.50%                 | 93.15%                     | 90.10%             | 90.58%                 | 0.65%                    | 0.48%                  |
| random        | const                  | const   | 92.51%                 | 93.07%                     | 89.88%             | 90.40%                 | 0.56%                    | 0.52%                  |
| random        | const                  | random  | 92.20%                 | 92.99%                     | 89.62%             | 90.25%                 | 0.79%                    | 0.63%                  |
| random        | random                 | const   | 92.47%                 | 93.35%                     | 89.78%             | 90.51%                 | 0.88%                    | 0.73%                  |
| random        | random                 | random  | 92.49%                 | 93.15%                     | 89.84%             | 90.65%                 | 0.66%                    | 0.81%                  |
| random        | random                 | random  | 92.65%                 | 93.57%                     | 89.98%             | 90.87%                 | 0.92%                    | 0.90%                  |
and $B_E$ can be defined as follows (compare with equation 1):

\begin{align*}
B_L &= \frac{1}{N} \sum_{i=1}^{N} f_B(\theta_{B_i}, f_C(O, f_T(\theta_T, O))) \\
B_E &= \frac{1}{N} \sum_{i=1}^{N} f_B(\theta_{B_i}, f_C(O, 1/2 \sum_{j=1}^{N} f_T(\theta_{T_j}, O_E)))) \\
O_E &= \frac{1}{N} \sum_{k=1}^{N} f_O(I, f_P(\theta_{P_k}, I, c))
\end{align*}

(12)

Three Frustum PointNet instances were trained for both $v1$ and $v2$ configurations. Accuracy is calculated for ground and 3D detection. Results are limited by the 2D bounding box proposal accuracy. Objects are divided into three classes (car, pedestrian and cyclist) and three difficulty levels (easy, medium and hard). Tables 6 and 7 presents the accuracy for object classes and for difficulty levels, respectively.

Note that configuration $v1$ (based on PointNet) often outperforms $v2$ (based on PointNet++) when ensemble is not used, whereas with ensemble of 3 model instances, $v2$ is better. The performance gain is also higher than that for ModelNet40 classification, which suggests that training set size is smaller concerning the complexity of the task. Averaging outputs of all modules outperforms averaging of only last module output.

### 3.7 Comparison of Computational Runtime

We benchmark the speed of all the architectures on the Jetson TX2 platform. We chose a mini-batch of four point clouds because it seemed to be a reasonable amount of segmented objects visible in a typical mobile robot environment. One has to keep in mind that we have not performed any target-specific optimization. All approaches used (NVIDIA CUDA) acceleration and three different deep learning frameworks, based on the original authors’ implementations of these methods.

As depicted in Figure 7, DeepSets has the smallest number of parameters and is significantly faster than that of the other approaches. Surprisingly, PointNet has considerable amount of parameters but is also pretty fast. It is worth pointing out that an increase in a few percentages of classification accuracy (i.e. SO-Net or PointCNN) is occupied by significantly longer execution times.

### 4 Conclusion

In this article, we focus on the examination of ensemble learning on 3D point cloud classification with seven most popular architectures using raw point sets. We examine the possibility to leverage the classification accuracy of

| Model                  | Car   | Pedestrian | Cyclist |
|------------------------|-------|------------|---------|
| 2D proposal            | 91.5  | 77.8       | 77.4    |
| Accuracy ground        |       |            |         |
| $v1$, no ensemble      | 81.7  | 62.6       | 63.8    |
| $v1$, last module ensemble | 82.6  | 64.4       | 65.5    |
| $v1$, all modules ensemble | 82.9  | 65.9       | 66.9    |
| $v2$, no ensemble      | **82.3** | **63.3**  | **65.0** |
| $v2$, last module ensemble | 83.3  | 65.7       | 67.8    |
| $v2$, all modules ensemble | **83.6** | **66.5**  | **68.3** |
| Accuracy 3D            |       |            |         |
| $v1$, no ensemble      | 72.4  | 58.1       | 58.9    |
| $v1$, last module ensemble | 74.1  | 59.4       | 61.5    |
| $v1$, all modules ensemble | **74.4** | 60.1       | 62.9    |
| $v2$, no ensemble      | 72.0  | **56.5**   | **60.8** |
| $v2$, last module ensemble | 73.6  | 59.6       | 63.7    |
| $v2$, all modules ensemble | 74.0  | **60.5**   | **65.0** |
DeepSets has the smallest number of parameters and is significantly faster than that of the other approaches.

Each of the seven cited models by ensemble learning. First, we observe which voting policy is the best for the task. Second, we found slightly better classification accuracy with the increasing number of models in ensemble along with smaller standard deviation. It proves that the ensemble’s output is more stable and reliable. The biggest mean instance classification accuracy gain was observed for KCNet — (2.52%), SO-Net — (0.99%), and other for architectures — (0.50%) on average. Significant increase in classification accuracy achieved by KCNet in comparison with all other architectures could be caused by different underlying working principle of kernel correlation as a measure of neuron activation. Or could suggest there is some more space for hyperparameters tweaks in KCNet, e.g. the number of filters (sets of kernels).

One can see some interesting observations due to the comparative study of different models. For example, there are classes where the simplest (global) approaches (PointNet, DeepSets) present the best classification accuracy. This suggests that the general shape of some objects is more important than the shape of local structures, which could mislead more complex models.

We also showed that the ensemble models with different architectures can further leverage the overall accuracy. We found that the SO-Net got the highest instance and mean class classification accuracy, but PointCNN wins in the number of classes in which given network obtained the highest accuracy after ensemble (this score is also high for DGCNN). This suggests that the latter can be much better in classifying some classes and much worse in other ones. This could also explain why the ensemble of only two model instances with different architectures lead to approximately 1.5% of instance classification accuracy increase compared to the state of the art results. This increase is equal to one obtained

Table 7: Frustum PointNet ensemble results for difficulty levels.

| Model               | Easy | Medium | Hard | All  |
|---------------------|------|--------|------|------|
| 2D proposal         | 90.0 | 79.8   | 76.9 | 82.2 |
| Accuracy ground     |      |        |      |      |
| v1, no ensemble     | 79.1 | 67.6   | 61.5 | 69.4 |
| v1, last module ensemble | 80.3 | 69.0   | 63.2 | 70.8 |
| v1, all modules ensemble | 81.3 | 70.6   | 63.9 | 71.9 |
| v2, no ensemble     | **78.8** | **68.9** | **63.0** | **70.2** |
| v2, last module ensemble | 80.8 | 71.2   | 64.8 | 72.3 |
| v2, all modules ensemble | **81.3** | **71.8** | **65.3** | **72.8** |
| Accuracy 3D         |      |        |      |      |
| v1, no ensemble     | 74.3 | 60.5   | 54.5 | 63.1 |
| v1, last module ensemble | 76.4 | 62.4   | 56.1 | 65.0 |
| v1, all modules ensemble | 77.3 | 63.2   | 56.9 | 65.8 |
| v2, no ensemble     | **73.5** | **60.8** | **54.9** | **63.1** |
| v2, last module ensemble | 76.6 | 63.2   | 57.2 | 65.6 |
| v2, all modules ensemble | **77.5** | **64.0** | **58.2** | **66.5** |
while using 10 instances of SO-Net model. Further gain can be achieved while using multiple model instances for each of mixed architectures. Instance accuracy of 94.03% can be obtained for two architectures and 94.15% while combining three or four architectures.

The performance gain is even higher for Frustum PointNet’ architecture evaluated on real-world KITTI dataset, using only three model instances.

We tested the source of randomness in ensemble learning analysis for SO-Net. We observed that numerical issues of massively parallel computations in deep neural networks are essential and beneficial in ensemble learning. The ensemble of several classifiers with the same encoder does not result in a significant performance gain. Simple ensemble outperforms classic bagging for tested approaches.

In addition we provide some tips for implementing point cloud classification into a mobile robot equipped with the Jetson TX2 platform by comparing inference time for all the tested models.

There are more questions one can ask around the topic of ensemble learning for point cloud processing. In our opinion, results of this study could leverage the benefit of knowledge distillation in real-world 3D object detection for autonomous cars and mobile robots.

References

[1] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. International Journal of Robotics Research (IJRR), 2013.
[2] Riyad A El-laithy, Jidong Huang, and Michael Yeh. Study on the use of Microsoft Kinect for robotics applications. In Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION, pages 1280–1288. IEEE, 2012.
[3] Michael Himmelsbach, Thorsten Luettel, and H-J Wuensch. Real-time object classification in 3D point clouds using point feature histograms. In Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, pages 994–1000. IEEE, 2009.
[4] Martin Rutzinger, Bernhard Höfle, Markus Hollaus, and Norbert Pfeifer. Object-Based Point Cloud Analysis of Full-Waveform Airborne Laser Scanning Data for Urban Vegetation Classification. Sensors, 8(8):4505–4528, 2008.
[5] Daniel Munoz, Nicolas Vandapel, Martial Hebert, et al. Directional Associative Markov Network for 3-D Point Cloud Classification. In Fourth international symposium on 3D data processing, visualization and transmission, pages 65–72. Georgia Institute of Technology Atlanta, US, 2008.
[6] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D Convolutional Neural Network for real-time object recognition. In Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, pages 922–928. IEEE, 2015.
[7] Konstantinos Sfikas, Ioannis Pratikakis, and Theoharis Theoharis. Ensemble of PANORAMA-based convolutional neural networks for 3D model classification and retrieval. Computers & Graphics, 71:208–218, 2018.
[8] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 1(2):4, 2017.
[9] Shuran Song and Jianxiong Xiao. Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 808–816, 2016.
[10] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. Deep Learning, volume 1. MIT press Cambridge, 2016.
[11] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1912–1920, June 2015.
[12] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In Proceedings of the IEEE international conference on computer vision, pages 945–953, 2015.
[13] Kripasindhu Sarkar, Basavaraj Hampiholi, Kiran Varanasi, and Didier Stricker. Learning 3D Shapes as Multi-Layered Height-maps using 2D Convolutional Networks. arXiv preprint arXiv:1807.08485, 2018.
[14] Barnabas Poczos, Aarti Singh, Alessandro Rinaldo, and Larry Wasserman. Distribution-free distribution regression. In Carlos M. Carvalho and Pradeep Ravikumar, editors, Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, volume 31 of Proceedings of Machine Learning Research, pages 507–515, Scottsdale, Arizona, USA, 29 Apr–01 May 2013. PMLR.
[15] In-Soo Jung, Mario Berges, James H. Garrett, and Barnabas Poczos. Exploration and evaluation of AR, MPCA and KL anomaly detection techniques to embankment dam piezometer data. *Advanced Engineering Informatics*, 29(4):902 – 917, 2015.

[16] M. Ntampaka, H. Trac, D. J. Sutherland, S. Fromenteau, B. Póczos, and J. Schneider. Dynamical mass measurements of contaminated galaxy clusters using machine learning. *The Astrophysical Journal*, 831(2):135, 2016.

[17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017.

[18] Jiaxin Li, Ben M Chen, and Gim Hee Lee. SO-Net: Self-Organizing Network for Point Cloud Analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9397–9406, 2018.

[19] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining Point Cloud Local Structures by Kernel Correlation and Graph Pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 4, 2018.

[20] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

[21] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[22] Shijie Lin, Jinwang Wang, Wen Yang, and Gui-Song Xia. Toward Autonomous Rotation-Aware Unmanned Aerial Grasping. *CoRR*, abs/1811.03921, 2018.

[23] Jie Tang, Yong Ren, and Shaoshan Liu. Real-Time Robot Localization, Vision, and Speech Recognition on Nvidia Jetson TX1. *CoRR*, abs/1705.10945, 2017.

[24] Jong-Chyi Su, Matheus Gadelha, Rui Wang, and Subhransu Maji. A Deeper Look at 3D Shape Classifiers. *CoRR*, abs/1809.02560, 2018.

[25] Varun Arvind, Anthony Costa, Marcus Badgeley, Samuel Cho, and Eric Oermann. Wide and deep volumetric residual networks for volumetric image classification. *arXiv preprint arXiv:1710.01217*, 2017.

[26] Charles Ruizhongtai Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. *CoRR*, abs/1809.02560, 2018.