EPiC: Ensemble of Partial Point Clouds for Robust Classification

Meir Yossef Levi, Guy Gilboa
Viterbi Faculty of Electrical and Computer Engineering
Technion - Israel Institute of Technology, Haifa, Israel
me.levi@campus.technion.ac.il ; guy.gilboa@ee.technion.ac.il

Abstract

Robust point cloud classification is crucial for real-world applications, as consumer-type 3D sensors often yield partial and noisy data, degraded by various artifacts. In this work we propose a general ensemble framework, based on partial point cloud sampling. Each ensemble member is exposed to only partial input data. Three sampling strategies are used jointly, two local ones, based on patches and curves, and a global one of random sampling.

We demonstrate the robustness of our method to various local and global degradations. We show that our framework significantly improves the robustness of top classification networks by a large margin. Our experimental setting uses the recently introduced ModelNet-C database by Ren et al. [24], where we reach SOTA both on unaugmented and on augmented data. Our unaugmented mean Corruption Error (mCE) is 0.64 (current SOTA is 0.86) and 0.50 for augmented data (current SOTA is 0.57). We analyze and explain these remarkable results through diversity analysis. Our code is available at: https://github.com/yossilevii100/EPiC

1. Introduction

A major obstacle in data-driven algorithms is the strong relation between performance and precise knowledge of input statistics. A way to test network robustness is to create corrupted test sets, where training is unaware of the specific corruptions. In recent years this has been investigated for images, with the creation of corrupted benchmarks e.g.: ImageNet-C, CIFAR10-C, CIFAR100-C [11] and MNIST-C [17]. In [24] the idea is extended to point cloud classification, with the introduction of ModelNet-C. In our work we present a generic framework, based on sampling, which is light, flexible and robust to Out-Of-Distribution (OOD) samples.

Ensemble learning is a long-standing concept in robust machine learning [8, 10, 6]. We would like to obtain an ensemble of learners, which are loosely correlated, that generalize well also to OOD input. There are two main questions: 1) How to form these learners? 2) How to combine their results?

One classical way to form ensembles in deep learning (see for instance [2]), is to rely on the partial-randomness of the training process. In this setting, the ensemble consists of networks of the same architecture trained on the same training-set. Variations of networks’ output stem from the different random parameter initialization and the stochastic gradient descent process. This induces Stochastic diversity. Another major approach to generate ensembles (both in classical machine learning and in deep learning [8, 2, 39]) is to change the sampling distribution of the training set for each learner. A mixture-of-experts approach is to train different types of classifiers, in neural networks this is accomplished by different network architectures. This induces

(a) Random (b) Patch (c) Curve

Figure 1: EPiC concept. Three sampling mechanism are used in our ensemble: Random captures global information, Patch holds full local resolution, and Curve is more exploratory in nature. Blue - anchor point, Red - sampled points. We show and explain why such ensembles are highly robust to various corruptions.
Architecture diversity. These approaches, however, may not yield sufficient OOD robustness (our experiments show their diversity is limited). Deep ensembles are being investigated with new ways to calibrate and to estimate uncertainty through increasing the ensemble diversity [38, 30].

Our approach to form ensembles is by exposing each ensemble member to limited input data. It is performed by generating different samples of each point cloud at both training and testing, see Fig. 1. The ensemble becomes highly diverse, since each classifier has access to different parts of the data. Moreover, each sampling method has unique characteristics. We observe that many types of corruption do not corrupt the data uniformly (see examples in Fig. 2). Some partial point clouds may thus be less corrupted, achieving accuracy closer to the case of clean data. Highly corrupted partial point clouds yield almost random network response, and can be modeled as adding noise to the classification decision. In this case, applying mean over the outputs of the ensemble significantly diminishes noise and improves accuracy. Four major advantages are gained by our proposed scheme: 1) The framework is completely generic and can essentially work with any point cloud classification network (we experimented with five, gaining improvement in all); 2) Many diverse classifiers can be generated; 3) Partial data is robust to local corruptions and to outliers, performing well on OOD samples; 4) The required networks to be trained is the number of sampling methods (in our case three), and not the ensemble size.

We reach robustness to various local and global degradations, yielding state-of-the-art results on corrupted ModelNet-C [24] ($mCE = 0.646$ using PCT [9] and $mCE = 0.501$ using augmented with WolfMix [25, 4] version of RPC [24]), even with a small ensemble of size 12.

2. Related Work

Point Cloud classification. It is customary to categorize point cloud classification networks to three mechanisms: multi-view, voxelizing, and point-wise networks. One of the challenges in point cloud processing is that, unlike images, 3D points are irregular in space. Multi-view methods project the point cloud into different viewpoints, generate 2D images, and apply CNN based networks [22, 1, 27]. The projection forces the data to be regular. These approaches are slower because of the rendering phase, and might lose useful geometric information. In the voxelizing mechanism, the solution for the irregularity problem is to partition the 3D space into a voxel grid [22, 32, 16]. This approach suffers heavily from sensitivity to the choice of grid regularity. In the context of point-wise networks, PointNet [20] presented a pioneering method of applying MLP on the raw 3D points. DGCNN [31], Dynamic Graph CNN, dynamically constructs a graph through the network. The graph is initially based on the raw 3D points, and progresses to more evolved feature spaces with semantic connectivity. GDANet [13], Geometry-Disentangled Attention Network, dynamically disentangles point clouds into contour and flat components. Respective features are fused to provide distinct and complementary geometric information from each representation. Recently, PCT [9], Point Cloud Transformer, adopted transformer architecture [29] to create per-patch embedding, leveraging self-attention mechanisms. CurveNet [26] generates features for each point by guided-walk over the cloud followed by a curve grouping operator. Ren et al. [24] studied the impact of several architectural choices on the robustness of the network and combined the most robust components to create RPC, which is the SOTA network on ModelNet-C. RPC takes advantage of 3D representations, using KNN, frequency grouping and self-attention mechanism that turned out to be the most robust combination. Another related work is Point-BERT [36] which splits the point cloud into patches as well. It then tokenizes the patches and uses pre-trained transformers with Mask Point Modeling. Point-MLP [15] is based on residual MLPs, achieving impressive results on the clean data.

Robustness to shifts from the training set. Traditionally, most of the attention in terms of robustness focused on basic deformations: jitter, scale and rotation. There are different approaches in the literature focusing on some specific deformations, degradations or corruptions. For example, a long standing research topic is rotation invariance, thoroughly addressed, e.g. by ClusterNet [3] and LGR-Net [7]. PointCleanNet [23] and PointASNL [35] addressed specifically robustness to outliers. Studies investigating robust-
### Table 1: Main Experimental Result. ModelNet-C Unaugmented classification comparison. Bold best, underline second best. Our framework dramatically improves robustness of all examined networks, as indicated by the mCE measure. Experiments using EPiC were conducted on the five most robust networks ($mCE \leq 1$).

<table>
<thead>
<tr>
<th>Network (#Ensemble size)</th>
<th>OA ↑</th>
<th>mCE ↓</th>
<th>Scale</th>
<th>Jitter</th>
<th>Drop-G</th>
<th>Drop-L</th>
<th>Add-G</th>
<th>Add-L</th>
<th>Rotate</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGCNN [31]</td>
<td>92.6%</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>EPiC(#12) (Ours)</td>
<td>93.0%</td>
<td>0.669</td>
<td>1.000</td>
<td>0.680</td>
<td>0.331</td>
<td>0.498</td>
<td>0.349</td>
<td>0.807</td>
<td>1.019</td>
</tr>
<tr>
<td>GDANet [13]</td>
<td>93.4%</td>
<td>0.892</td>
<td><strong>0.830</strong></td>
<td>0.839</td>
<td>0.794</td>
<td>0.894</td>
<td>0.871</td>
<td>1.036</td>
<td>0.981</td>
</tr>
<tr>
<td>EPiC(#12) (Ours)</td>
<td>93.6%</td>
<td>0.704</td>
<td>0.936</td>
<td>0.864</td>
<td>0.315</td>
<td>0.478</td>
<td>0.295</td>
<td>0.862</td>
<td>1.177</td>
</tr>
<tr>
<td>CurveNet [26]</td>
<td>93.8%</td>
<td><strong>0.646</strong></td>
<td>0.894</td>
<td>0.851</td>
<td><strong>0.306</strong></td>
<td><strong>0.435</strong></td>
<td>0.285</td>
<td>0.735</td>
<td>1.019</td>
</tr>
<tr>
<td>EPiC(#12) (Ours)</td>
<td>93.0%</td>
<td>0.925</td>
<td>0.872</td>
<td>0.870</td>
<td>0.617</td>
<td>0.894</td>
<td>0.851</td>
<td>0.306</td>
<td>0.435</td>
</tr>
<tr>
<td>PCT [9]</td>
<td>93.4%</td>
<td>0.863</td>
<td>0.840</td>
<td>0.892</td>
<td>0.492</td>
<td>0.797</td>
<td>0.929</td>
<td>1.011</td>
<td>1.079</td>
</tr>
<tr>
<td>EPiC(#12) (Ours)</td>
<td>93.6%</td>
<td>0.750</td>
<td>0.915</td>
<td>1.057</td>
<td>0.323</td>
<td>0.440</td>
<td><strong>0.281</strong></td>
<td>0.902</td>
<td>1.330</td>
</tr>
<tr>
<td>PointNet [20]</td>
<td>90.7%</td>
<td>1.422</td>
<td>1.266</td>
<td>0.642</td>
<td>0.500</td>
<td>1.072</td>
<td>2.980</td>
<td>1.593</td>
<td>1.902</td>
</tr>
<tr>
<td>PointNet++ [21]</td>
<td>93.0%</td>
<td>1.072</td>
<td>0.872</td>
<td>1.177</td>
<td>0.641</td>
<td>1.802</td>
<td>0.614</td>
<td>0.993</td>
<td>1.405</td>
</tr>
<tr>
<td>RSCNN [34]</td>
<td>92.3%</td>
<td>1.130</td>
<td>1.074</td>
<td>1.171</td>
<td>0.806</td>
<td>1.517</td>
<td>0.712</td>
<td>1.153</td>
<td>1.479</td>
</tr>
<tr>
<td>SimpleView [1]</td>
<td><strong>93.9%</strong></td>
<td>1.047</td>
<td>0.872</td>
<td>0.715</td>
<td>1.242</td>
<td>1.357</td>
<td>0.983</td>
<td>0.844</td>
<td>1.316</td>
</tr>
<tr>
<td>PAConv [14]</td>
<td>93.6%</td>
<td>1.104</td>
<td>0.904</td>
<td>1.465</td>
<td>1.000</td>
<td>1.005</td>
<td>1.085</td>
<td>1.298</td>
<td>0.967</td>
</tr>
</tbody>
</table>

A major approach to increase robustness is by augmentation techniques which enrich the training set. RSmix [4] suggests mixing two samples from the database in a smooth manner, along with smoothing the labels of both samples, creating new virtual samples. This method inherently inserts jittering, scaling, translation and rotation. PointWolf [25] applies non-rigid manipulations on selected anchor points. These include scaling, 3D rotation and translation with local jittering. Such complex manipulations are able, for example, to change a person’s posture or the style of a chair. We note that augmentation introduces expected corruptions within the training set, thus violating, to some degree, the testing principles of OOD robustness. To make our work complete, we examine the proposed framework also on augmented data (WolfMix [24], a combination of PointWolf and RSmix). We achieve SOTA in this case as well.

### 2.1. Benchmarks

ModelNet-40 [32] is a widely used dataset consisting of CAD meshes from 40 classes, such as airplane, chair and sofa. Each mesh was uniformly sampled to form a 3D point cloud consisting of 1024 points. The dataset has 12,311 samples, divided to 9843 for training and 2468 for test. The dataset contains closely semantic classes like vase, plant and flower pot, or chair and stool, which makes this dataset highly challenging. Recently, [24] Ren et al proposed a corrupted point cloud benchmark to assess OOD robustness. It is based on the well studied ModelNet-40 and is referred to as ModelNet-C. We evaluate the robustness for OOD corruptions by applying it on ModelNet-C [24]. This dataset contains seven types of corruptions: jitter, scale, rotate, add-global, add-local, drop-global and drop-local. Each of these corruptions is rendered in five levels of difficulty. In [24] a unified calculation mechanism was defined to measure robustness. This measure is termed mCE (mean Corruption Error). It basically evaluates the results in comparison to DGCNN, which serves as a reference algorithm (and hence, by definition, has $mCE \equiv 1$). Each specific corruption has a similar score, relative to DGCNN, averaged over all five levels of difficulty. Since the measure is of error, a lower score is better. Please refer to [24] for more details.

### 3. Our Proposed Method

#### 3.1. Notations

We denote by $N$ the number of cloud points at the input of a network (which, due to sampling, may vary). The
number of features for each point in the cloud is $F$. We refer to $K$ as the ensemble size of each sub-sample mechanism and to $\tilde{K}$ as the combined ensemble size. In our case we suggest three sampling mechanisms of the same size, therefore $K = 3\tilde{K}$. $N_p$, $N_c$ and $N_r$ are the number of points in patches, curves and random sub-samples, respectively. In curve sampling there may be occasional repeating points, $N \leq N_c$. For curve extraction we have a hyper-parameter $M$ controlling the number of neighbors to choose from in a random-walk iteration. $C$ is the number of classes ($C = 40$ in ModelNet-40), and $p \in \mathbb{R}^C$ is a single prediction vector. $P \in \mathbb{R}^{K \times C}$ is an ensemble of predictions.

3.2. Motivation of our approach

The ability to classify in a robust manner is closely coupled with the ability to classify based on partial information. We would also like a classification network to perform reasonably well also when some information is missing, noise is added or when outliers are introduced. Thus, it is desired to obtain a diverse set of features which are based on different locations in the shape. This could be demonstrated well in the experiment illustrated in Fig. 3.

We visualize the internal classification importance of points for networks specializing in curves, patches and random. For the demonstration we focused on two degradations of adding and removing points locally. Commonly, a network gets a point cloud $X \in \mathbb{R}^{N \times 3}$ and encodes it into features $X_f \in \mathbb{R}^{N \times F}$ by a variety of sophisticated layers. The standard method to aggregate the points axis is by applying a symmetric function which is permutation invariant, such as max or mean. In this example, in order to obtain features $X_f$, we use DGCNN[31]. We calculate the importance of each point $j$, denoted by $Imp(j)$, in the following manner,

$$Imp(j) = \sum_{k=1}^{F} I (j == \text{arg max}_n (X_f(n,k))),$$

where $I$ is an indicator function (1 when true and 0 otherwise). The importance attempts to quantify the number of features of a specific point which are part of the global feature vector. This usually means that the feature’s magnitude at that point is maximal, dominating the respective feature vector entries among all other points. In this example, to cover the entire point cloud, we calculate the importance for all partial point clouds and present the average result in Fig. 3.

When parts of the cloud are missing, or outliers are introduced, potentially corrupting features, we want the classification to be based on a variety of features, even though they may be less prominent in the clean set. Therefore, our aim is to “spread” the importance as evenly as possible. This insight motivates our approach for using partial point cloud ensembles to impose feature diversity. Additional analysis and insights appear hereafter.

3.3. Proposed approach

We use three types of sub-samples: two local ones, Curves and Patches and a global one consisting of Random-sampling. For the local methods, we first use farthest point sampling (FPS) algorithm [21] to choose $\tilde{K}$ anchors. From each anchor we extract a patch and a curve.

Patch extraction is done by finding $N_p$ nearest neighbors. This sub-sample mechanism is more conservative, hence it preserves well the local information.
Curve extraction is done by a random-walk process, beginning from the anchor point. \( N_r \) random-walk iterations are performed, at each iteration one of \( M \) nearest neighbors is chosen randomly. The choice is with replacement (hence the sampled partial point cloud may be smaller than \( N_c \)). This mechanism is more exploratory in nature and less structured.

Random extraction is done by simply sub-sampling \( N_r \) random points from the entire point cloud (without replacement, \( N = N_r \)).

The values of these parameters were determined once and were used in the same manner in all our experiments for all classification networks (see details in Supp).

Generic framework. Our approach is generic and can be applied in conjunction with any point cloud classification network. The experiments (detailed in the experimental section) are conducted with five different architectures. These architectures are the most OOD-robust. Our method considerably improves the robustness of every one of them, as indicated by the mCE measure. We use three instances of the same architecture. Each instance is trained to classify point clouds obtained by a certain sub-sampling mechanism. A recap of our approach (inference) is given in Algorithm 1, where the training procedure is detailed in the Supp.

Algorithm 1 Classification using EPiC (inference)

```
Require: \( X, \text{params}_{\text{patches}}, \text{params}_{\text{Curves}}, \text{params}_{\text{Random}} \)
model\_patches \( \leftarrow \) params\_patches
model\_Curves \( \leftarrow \) params\_Curves
model\_Random \( \leftarrow \) params\_Random
anchors \( \leftarrow \) FarthestPointSampling\((X, K)\)
for \( k \in K \) do
    Patch \( \leftarrow \) FetchPatch\((X, \text{anchors}(k))\) \( \triangleright \) Local
    Curve \( \leftarrow \) FetchCurve\((X, \text{anchors}(k))\) \( \triangleright \) Local
    Random \( \leftarrow \) FetchRandom\((X)\) \( \triangleright \) Global
    \( P_{\text{Patch}}^k \leftarrow \) model\_patches\((\text{Patch})\)
    \( P_{\text{Curves}}^k \leftarrow \) model\_Curves\((\text{Curve})\)
    \( P_{\text{Random}}^k \leftarrow \) model\_Random\((\text{Random})\)
end for
\( P_{\text{ensemble}} \leftarrow \) Concatenate\((P_{\text{Patch}}^1:K, P_{\text{Curves}}^1:K, P_{\text{Random}}^1:K)\)
\( P \leftarrow \) Mean\((P_{\text{ensemble}})\); Class \( = \) arg max\((P)\).
```

4. Diversity Analysis

In order to leverage the advantage afforded by ensembles, a high level of diversity of ensemble members is required. This motivates us to quantitatively study the diversity of several types of ensembles. We investigate the following sources of diversity:

1. Stochastic. Diversity stemming from the stochasticity of the training process (initialization and SGD).

2. Architecture. Diversity caused by using different network architectures.

3. Sampling. Diversity due to different sampling methods and randomness of the sampling process.

Four types of ensembles were examined (each consisting of 12 members, correlation results are shown in Fig. 4):

1. No sampling, single architecture (NS-1A). Stochastic diversity: The ensemble consists of 12 instances of DGCNN[31]. The entire point-cloud is used as input.

2. No sampling, three architectures (NS-3A). Architecture + stochastic diversity. The ensemble consists of three architectures (each of 4 instances, in this order): PCT[9], GDANet[13] and DGCNN[31]. The entire point-cloud is used as input.

3. Sampling (ours), single architecture (S-1A). Sampling diversity. An ensemble consisting of three instances of DGCNN[31]. Each instance is trained to specialize in a different sampling mechanism. The sampling methods (in this order) are: Patches, Curves and Random. Each instance uses 4 different sub-sample inputs.

4. Sampling (ours), three architecture (S-3A). Sampling + architecture diversity. Similar setting to the sampling diversity experiment. Here the following architectures, GDANet[13], PCT[9] and DGCNN[31].

Figure 4: Correlation between ensemble members. Correlation output of ensemble members on full non-sampled clean test-set (NS), top, with a single and three architectures, compared to sampling (S) by our approach, bottom, see setting details in Section 4. \( c \) is defined in Eq. (2) (lower means higher diversity). Sampling affords higher diversity, compared to stochastic and architecture sources of diversity.
second best. Robustness improved by our suggested method.

were used on sampled inputs of Patches, Curves and Random, respectively.

In addition, PointGuard [12] was examined, as a different ensemble reference, with an ensemble of size 1000.

Measuring diversity through correlation. We introduce a measure which quantifies (inverse) diversification by

$$c = \frac{1}{S} \sum_{i=1}^{S} \frac{||C_i - I||^2}{K^2 - K},$$

where $S$ is the dataset size, $K$ is the ensemble size, $I_{K \times K}$ is a unit matrix, $|| \cdot ||$ is the Frobenius norm and $C$ is the Pearson correlation matrix of the ensemble predictions. The final measure $c \in [0, 1]$ is a scalar quantifying the diversity of the ensemble (in an inverse manner) for a given dataset. For $c = 0$ the members’ response is completely uncorrelated (most diverse). For $c = 1$ the members are fully correlated (zero diversity) and an ensemble is not necessary (would produce identical results as a single member). Note that since each member is quite accurate (a “strong learner”), with an accuracy of around 90%, on binary problems we expect a diverse ensemble to be with $c \approx 0.92$ (in the multiclass case analysis requires additional assumptions, but should be in similar ranges). As can be seen in Fig. 4, ensembles based on sampling are considerably more diverse than those based on stochastic or architecture diversity. Moreover, curves and patches are lowly correlated, although the same anchor points are used. Diversity stems mostly from the sampling scheme, in addition to locality. In Table 3 the improved robustness gained by the four ensemble methods is shown. Our partial-point-cloud strategy improves robustness in most criteria, excelling in mCE, with the three architecture configuration (S-3A) having superior results also in overall accuracy (clean set).

5. Experiments

We present our results for point cloud classification on ModelNet-C dataset, training on the clean dataset only and measuring performance on both clean and corrupted sets. Implementation details. We train the basic models independently on partial point clouds. The predictions are aggregated using mean. For the unaugmented version we use only basic, standard augmentation procedures (detailed below) in order not to violate the OOD principle. For WolfMix augmented version we first augment the entire sample, then we generate the different sub-samples. We eliminate the randomness with a fixed seed. All three models are trained simultaneously 300 epochs with learning rate of $1 \times 10^{-4}$, with cosine annealing scheduler [18] to zero. We use a batch size of 256. For the unaugmented version we followed DGCNN [31] protocol for augmentation: 1) random anisotropic scaling in the range $[2/3, 3/2]$; 2) random translation in the range $[-0.2, +0.2]$. The implementation uses Pytorch library [19]. Cross-Entropy loss is minimized. In the training phase we split each sample to 4 farthest point samples, and independently predict each partial point cloud class. The

<table>
<thead>
<tr>
<th>Networks (#Ensemble size)</th>
<th>OA↑</th>
<th>mCE↓</th>
<th>Scale</th>
<th>Jitter</th>
<th>Drop-G</th>
<th>Drop-L</th>
<th>Add-G</th>
<th>Add-L</th>
<th>Rotate</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGCNN[31]+W.M</td>
<td>93.2%</td>
<td>0.590</td>
<td>0.989</td>
<td>0.715</td>
<td>0.698</td>
<td>0.575</td>
<td>0.285</td>
<td>0.415</td>
<td>0.451</td>
</tr>
<tr>
<td>EPiC(#12)+W.M (Ours)</td>
<td>92.1%</td>
<td>0.529</td>
<td>1.021</td>
<td>0.541</td>
<td>0.355</td>
<td>0.488</td>
<td>0.288</td>
<td>0.407</td>
<td>0.600</td>
</tr>
<tr>
<td>GDANet[13]+W.M</td>
<td>93.4%</td>
<td>0.571</td>
<td>0.904</td>
<td>0.883</td>
<td>0.532</td>
<td>0.551</td>
<td>0.305</td>
<td>0.415</td>
<td>0.409</td>
</tr>
<tr>
<td>EPiC(#12)+W.M (Ours)</td>
<td>92.5%</td>
<td>0.530</td>
<td>0.968</td>
<td>0.639</td>
<td>0.343</td>
<td>0.473</td>
<td>0.275</td>
<td>0.433</td>
<td>0.577</td>
</tr>
<tr>
<td>PCT[9]+W.M</td>
<td>93.4%</td>
<td>0.574</td>
<td>1.000</td>
<td>0.854</td>
<td>0.379</td>
<td>0.493</td>
<td>0.298</td>
<td>0.505</td>
<td>0.488</td>
</tr>
<tr>
<td>EPiC(#12)+W.M (Ours)</td>
<td>92.7%</td>
<td>0.510</td>
<td>0.915</td>
<td>0.699</td>
<td>0.323</td>
<td>0.425</td>
<td>0.268</td>
<td>0.404</td>
<td>0.535</td>
</tr>
<tr>
<td>RPC[24]+W.M</td>
<td>93.3%</td>
<td>0.601</td>
<td>1.011</td>
<td>0.968</td>
<td>0.423</td>
<td>0.512</td>
<td>0.332</td>
<td>0.480</td>
<td>0.479</td>
</tr>
<tr>
<td>EPiC(#12)+W.M (Ours)</td>
<td>92.7%</td>
<td>0.501</td>
<td>0.915</td>
<td>0.680</td>
<td>0.315</td>
<td>0.420</td>
<td>0.251</td>
<td>0.382</td>
<td>0.544</td>
</tr>
</tbody>
</table>

Table 2: ModelNet-C Augmented (WolfMix) classification Comparison. Bold best, underline second best. 1) dramatically robustification gained by using EPiC on conventional methods. 2) Our method improves mCE and almost all OOD corruptions.

<table>
<thead>
<tr>
<th>Ensembles</th>
<th>OA↑</th>
<th>mCE↓</th>
<th>Scale</th>
<th>Jitter</th>
<th>Drop-G</th>
<th>Drop-L</th>
<th>Add-G</th>
<th>Add-L</th>
<th>Rotate</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS-1A [31]</td>
<td>93.5%</td>
<td>1.006</td>
<td>0.862</td>
<td>0.709</td>
<td>0.762</td>
<td>0.889</td>
<td>2.064</td>
<td>1.025</td>
<td>0.730</td>
</tr>
<tr>
<td>NS-3A [31, 9, 13]</td>
<td>93.4%</td>
<td>0.856</td>
<td>0.862</td>
<td>0.642</td>
<td>0.657</td>
<td>0.797</td>
<td>1.275</td>
<td>0.982</td>
<td>0.777</td>
</tr>
<tr>
<td>S-1A [31], Ours</td>
<td>93.0%</td>
<td>0.669</td>
<td>1.000</td>
<td>0.680</td>
<td>0.331</td>
<td>0.498</td>
<td>0.349</td>
<td>0.807</td>
<td>1.019</td>
</tr>
<tr>
<td>S-3A [31, 9, 13], Ours</td>
<td>93.8%</td>
<td>0.671</td>
<td>0.915</td>
<td>0.813</td>
<td>0.315</td>
<td>0.469</td>
<td>0.302</td>
<td>0.811</td>
<td>1.074</td>
</tr>
<tr>
<td>Point-Guard (#1,000) [12]</td>
<td>89.6%</td>
<td>0.949</td>
<td>1.947</td>
<td>0.617</td>
<td>0.427</td>
<td>0.599</td>
<td>0.376</td>
<td>0.702</td>
<td>1.977</td>
</tr>
</tbody>
</table>

5.1. Main evaluation results

The EPiC framework is implemented using several widely used point cloud classification networks. Table 1 shows the Unaugmented results and Table 2 shows results following augmentation by WolfMix [25, 4].

**Unaugmented.** A major advantage is achieved in terms of mCE for all networks. Using PCT[9] with EPiC reaches mCE=0.646, which outperforms current SOTA of RPC[24] (mCE=0.863) by a large margin. With respect to accuracy on the clean data set (OA), we see improvement in four out of five cases, with only CurveNet[26] degrading, where RPC[24] and GDANet[13] reach 93.6%.

**Augmented (WolfMix).** We further examined whether our method can improve augmentation procedures (which to some extent violate some OOD assumptions). EPiC on augmented data consistently improves robustness. Using RPC [24] EPiC achieves mCE=0.501 surpassing current augmented SOTA (mCE=0.571). OOD robustness comes at a cost of minor accuracy drops, consistently for all networks. The trade-off between accuracy and robustness is well demonstrated in [24] for point-cloud classification, and in [28] for general classification settings.

5.2. When our method performs best?

Let \( X \in \mathbb{R}^{N \times 3} \) be the set of coordinates of a clean point cloud. Following a corruption transformation \( T_c \) acting on \( X \) we get a corrupted point cloud \( X_c \in \mathbb{R}^{M \times 3} \), where \( X_c = T_c(X) \). Let the intersection of these sets be defined by \( X_\cap := X \cap X_c \) of size \( |X_\cap| \) points. We define the uniformity of the corruption transformation by

\[
u(X, T_c) := 1 - \frac{|X_\cap|}{\max(N, M)}.
\]

For a fully uniform corruption \( T_c \), all points of the original point cloud change, hence \( X_\cap = \emptyset \) and \( |X_\cap| = 0 \), yielding \( u = 1 \). When the transformation is highly selective, affecting only a few points, \( X_\cap \approx X \) and \( u \rightarrow 0 \). We refer to the latter as a highly nonuniform corruption. This measure is very general and can quantify various diverse corruptions.

We can roughly divide the corruptions of ModelNet-C to mostly not exposed to the corruption, as shown in Fig. 2. In these cases the classification is with high accuracy. Partial point clouds which are highly exposed to the corruption have high chances of yielding missclassification. However, our experiments indicate these missclassifications are quite random and can be approximately modeled as noise. As we perform a mean operation over the ensemble outputs, this noise is mostly averaged out, where the correct (mostly unexposed) members of the ensemble dominate the decision. This phenomenon can be seen in the experiment shown and explained in Fig. 6. A plot of mCE as a function of \( u \) is shown in Fig. 5, illustrating the above rationale.

**The case of Jitter.** Let us assume the points in the cloud are sampled approximately evenly, with a mean distance between each point of \( \ell \). Let the Jitter corruption be of standard deviation \( \sigma \). If \( \sigma \ll \ell \) essentially the point cloud is similar to the original clean one and most classifiers would perform well. In terms of the relaxed definition of uniformity, we can set \( \epsilon = \ell \) and for \( \sigma \ll \epsilon \) we get \( u \rightarrow 0 \) (the corruption is “invisible” to the classification network). Random sampling copes very well with Jitter. We can view random sampling as reducing the point cloud resolution. Basically, we now have a larger distance \( L > \ell \) between the points. Since the classifier is trained on the low resolution input, as long as \( \sigma \ll L \) we get \( u \rightarrow 0 \). Hence, we are more robust to Jitter with a larger standard deviation. See [12] for additional perspectives and insights on this topic.

![Figure 5: mCE versus uniformity.](image)

5.3. Sampling, aggregation and network size

In Table 4 we show experimental results for each subsampling method with DGCNN[31] as the basic model (additional models are shown in the Supp.). Ensembles of size four are tested, aggregated using mean. In addition, the results of all three methods (forming an ensemble of 12 members) are aggregated using either mean or majority-voting. Each sampling method has its strengths and weaknesses: Patches are much better in terms of accuracy on clean point clouds. In this case features are well preserved and patches.
Table 4: Sub-Samples vs. Aggregated. **Bold** best among aggregations, **underline** best among sub-samples. Aggregations are almost always superior for any corruption. Thus, it can be inferred that the partial classifiers are low-correlated.

<table>
<thead>
<tr>
<th>Sub-samples</th>
<th>OA ↑</th>
<th>mCE ↓</th>
<th>Scale</th>
<th>Jitter</th>
<th>Drop-G</th>
<th>Drop-L</th>
<th>Add-G</th>
<th>Add-L</th>
<th>Rotate</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGCNN-Curves (#4)</td>
<td>90.7%</td>
<td>1.069</td>
<td>1.628</td>
<td>1.297</td>
<td>0.431</td>
<td>0.729</td>
<td>0.363</td>
<td>1.618</td>
<td>1.414</td>
</tr>
<tr>
<td>DGCNN-Patches (#4)</td>
<td>92.8%</td>
<td>0.793</td>
<td><strong>0.989</strong></td>
<td>1.165</td>
<td>0.577</td>
<td>0.536</td>
<td>0.505</td>
<td>0.851</td>
<td><strong>0.930</strong></td>
</tr>
<tr>
<td>DGCNN-Random (#4)</td>
<td>91.5%</td>
<td>0.766</td>
<td>1.234</td>
<td><strong>0.399</strong></td>
<td>0.351</td>
<td>0.812</td>
<td>0.580</td>
<td><strong>0.793</strong></td>
<td>1.195</td>
</tr>
<tr>
<td>DGCNN-Mean (#12)</td>
<td><strong>93.0%</strong></td>
<td><strong>0.669</strong></td>
<td>1.000</td>
<td>0.680</td>
<td><strong>0.331</strong></td>
<td><strong>0.498</strong></td>
<td><strong>0.349</strong></td>
<td>0.807</td>
<td>1.019</td>
</tr>
<tr>
<td>DGCNN-Maj. Voting (#12)</td>
<td>92.6%</td>
<td>0.706</td>
<td>1.043</td>
<td>0.794</td>
<td>0.359</td>
<td>0.517</td>
<td>0.380</td>
<td>0.800</td>
<td>1.051</td>
</tr>
</tbody>
</table>

Table 5: Network size vs Performance. EPiC with DGCNN-v2 is an excellent compromise of a lean architecture, with a small number of parameters, which is much more robust (and almost as accurate), compared to the original full DGCNN network.

<table>
<thead>
<tr>
<th>Network</th>
<th>OA ↑</th>
<th>mCE ↓</th>
<th>#parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic DGCNN</td>
<td>92.6%</td>
<td>1.000</td>
<td>1.8M</td>
</tr>
<tr>
<td>EPiC based on DGCNN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DGCNN-v1</td>
<td>88.7%</td>
<td>1.041</td>
<td>159K</td>
</tr>
<tr>
<td>DGCNN-v2</td>
<td><strong>92.2%</strong></td>
<td><strong>0.773</strong></td>
<td><strong>636K</strong></td>
</tr>
<tr>
<td>DGCNN-v3</td>
<td>92.3%</td>
<td>0.720</td>
<td>2.47M</td>
</tr>
<tr>
<td>DGCNN-v4</td>
<td>93.0%</td>
<td>0.669</td>
<td>5.4M</td>
</tr>
<tr>
<td>DGCNN-v5</td>
<td><strong>92.7%</strong></td>
<td>0.684</td>
<td>39M</td>
</tr>
</tbody>
</table>

Figure 6: Exposure to corruption modeled as noise. This illustration is based on the following experiment: All instances of class Table were corrupted by Add-Local. Curve is used for sampling. The sampled instances were divided into two groups: Unexposed (left) are curves containing less than 10 corrupted points. Exposed (right) are curves containing more than 50 corrupted points. Soft-Max predictions of each group were averaged (bottom row). The prediction of exposed curves is almost random (highly noisy) and can be well handled by averaging over the ensemble outputs. Unexposed instances are classified well.

Affect of network size. Since each ensemble member has access only to partial data, we check whether a full size model is required. Five versions of DGCNN are examined, the most shallow is v1 and the deepest is v5, where v4 is the original network (architectures details appear in the Supp.). The results are shown in Table 5. The number of parameters required for the entire ensemble (3 instances) is shown on the right column. Note that v2 yields a robust version with about third of the parameters of the classical network (top row), with just a slight degradation in overall accuracy.

6. Conclusion

In this work, we demonstrated the OOD robustness of ensembles based on partial information input for the task of point cloud classification. The approach relies on obtaining lowly-correlated input samples for each member of the ensemble. We integrate three types of sampling schemes: Curves, Patches and Random. In terms of training - only a single network for each sampling scheme is trained, which saves training time. The ensemble is created in real time. This naturally increases inference time, but is reasonable for small ensembles (we show a size of $K = 12$). For highly demanding real time applications, the networks can be duplicated and processed in parallel. For highly demanding memory consumption applications, significantly smaller networks can be used, which are still highly robust. Note that ensembles can be distilled back to a single network, for better hardware and time efficiency, as suggested for in-
stance in [41]. Since our proposed approach is purely abstract it can be extended to additional problems. We plan to further investigate how such mechanisms can improve robustness in other fields as well.

Acknowledgements

We acknowledge support by the Israel Science Foundation (Grant No. 534/19), by the Ministry of Science and Technology (Grant No. 5074/22) and by the Ollendorff Minerva Center.

References

[24] Jiawei Ren, Liang Pan, and Ziwei Liu. Benchmarking and analyzing point cloud classification under corruptions. International Conference on Machine Learning (ICML), 2022. 1, 2, 3, 6, 7


14484