UNDER REVIEW

RESCIENCE C

Replication / ML Reproducibility Challenge 2022 / 3D Vision

Reproducibility study of "Surface Representation for Point Clouds"

İsmail Çetin[1] and Batuhan Cengiz[2]
[1,2]Istanbul Technical University

Edited by
(Editor)

Reviewed by
Abdullah Akgül, Fırat Öncel
(Reviewer 2)

Received
08 January 2023

Published
–

DOI
–

Reproducibility Summary

Scope of Reproducibility – In this work, we study the reproducibility of the paper Surface Representation for Point Clouds (RepSurf) by Haoxi Ran, Jun Liu and, Chengjie Wang. To verify their main claims, which state that with an increase of only a small amount of parameters and computational cost, the proposed model achieves (i) increased performance on the classification task by a large margin (ii) increased intersection ratio in Mean Intersection over Union (mIoU) for the segmentation task. We have conducted our experiments only on the classification task.

Methodology – The PyTorch training source codes have been provided by the authors of the paper for the classification and segmentation tasks. However, they have not provided any code for object detection. Implementations for some of the datasets are also missing in given tasks. Overall, we have focused on the classification task and rerun experiments with the given source code which cost around 13 hours of GPU time on a TITAN RTX GPU.

Results – We found that claims of paper in terms of increase to the performance and model complexity holds. We were able to reproduce the authors' classification results in task with the ScanObjectNN dataset. We reproduced the accuracy to within 1% of the reported value, which confirms the validity of the experiments.

What was easy – The original code implementation was easily accessible to the public, with intuitively named variables and functions that reflected their roles in the project. The research paper also provided a comprehensive appendix with further implementation details and pseudo-codes. It is easy to start with the experiments, because the implementation included configuration files, download scripts for necessary datasets and pretrained weights.

What was difficult – Firstly, even though code was well written in terms of variable naming, the connections between the function and the modules of the code was problematic and hard to grasp. Also, the dataset is provided in a processed form in which labels are encoded without their true object name. Also in the paper, in the appendix part they used *random dropout* keyword which is a tautology. Due to its nature, dropout is already random. Lastly, authors have written their code in a way to run in their multi-GPU

Correspondence should be addressed to İsmail Çetin, Batuhan Cengiz (cetini18@itu.edu.tr, cengiz16@itu.edu.tr)
The authors have declared that no competing interests exists.
Code is available at https://github.com/hancyran/RepSurf.

cluster without documenting the modifications to the environment variables. Therefore, we had difficulty to run the code in the first step with our environment.

Communication with original authors — We have not contacted with authors up to now. Briefly describe how much contact you had with the original authors (if any).

1 Introduction

Point cloud is a discrete set of data points, each of which is associated with three coordinate values and may contain other additional values like color. Learning from point clouds has a crucial importance in autonomous driving, manufacturing and for other industries as well. However, due to irregularity of point clouds, it is not straightforward to learn directly from them.
There has been previous works on point clouds, such as PointNet [1] which uses Multilayer Perceptron Networks (MLP) and PointNet++[2] which uses set abstraction with MLPs to capture local geometry.
Ran et. al. claims that[3], previous attempts has not been able to capture local geometry and these methods only learn global features with a symmetrical function, such as max operation. To learn from local features some indirect mechanism which are sophisticated, are devised and network learns implicitly through this mechanism. However, they assert that these mechanism may result in omission of information regarding the input, or a loss of geometry during transformation. An explicit local structure is needed and the aim of Representative Surfaces (RepSurf) is to provide this. The network (plug-and-play) module they provide, should increase the performance on the downstream tasks. There are three types of downstream tasks mentioned in the paper Classification, Segmentation and Object Detection namely.
In this reproducibility challenge, Our goal is to replicate the findings of the Ran et. al.[3], validate their assertions, and conduct experiments to provide visual evidence to support their claims. Through this effort, we aim to make the following contributions:

- We reproduce the classification experiments conducted by the authors to identify which parts of the experimental results supporting their claims can be reproduced, and at what cost in terms of resources and validate them
- We also compare the given code and techniques method in the paper, and check if they are logically the same and available to use.

2 Scope of reproducibility

Extracting semantic information from point cloud is an active research area which involves tasks such as classification, segmentation and object detection. Local representations are important factors when they are learned explicitly for extracting meaningful information. Representative Surfaces (Repsurf), inspired by Taylor series, is created to capture local information explicitly. Triangular Repsurf and Umbrella Repsurf are a type of Repsurf, the latter is the extension of the former. In this reproducibility study, our main goal is to verify the following claims of the original paper:

- **Explicitly Learning Local Information:** They claim that, they can represent local geometry with second-order derivative Taylor series centered on point (t, f(t)) of a 2D curve by Taylor series.
- **Increased Performance:** The authors claim that Umbrella RepSurf applied to PointNet++ (SSG 2x) architecture surpass state-of-the-art point cloud classification models by a large margin.
- **Minimum Computational Complexity:** Their third claim is that the increase in computational complexity is minimum and their method is simple to implement as a plug-and-play module.

The rest of this work is organized as follows. In Section 3, we outline our approach for reproducing the original paper. Then, in Section 4, we present the results of our replication results and compare them to the results presented in the original paper. Finally, in

Section 5, we reflect on our experience reproducing the research and discuss what we found.

3 Methodology

The original implementation of the paper is publicly available [4] for classification and segmentation tasks. In this reproducibility project, we have focused on classification task on the ScanObjectNN [5] dataset based on authors' implementation. We have conducted our experiments on a Nvidia RTX TITAN GPU. We have modified code and scripts minimally only to fit into our setup.

3.1 Model descriptions

We have run experiments for Umbrella Repsurf and also the Umbrella Repsurf x2 (Multi-scale Inference[6]). The original algorithm is also given in Figure 1.

Algorithm 2 Pytorch-Style Pseudocode of **Umbrella RepSurf**

```
# B: batch size, N: number of points
# K: number of neighbors, C: output channels
# points: coordinates of a point set
neighbors = kNN(points, k=K)-points # [B,N,K,3]
edges = sort_by_clock(neighbors) # [B,N,K,3]
edges = unsqueeze(neighbors, dim=-2) # [B,N,K,1,3]
pairs = concat([edges, edges.roll(-1, 2)], dim=-2)
      # [B,N,K,2,3]
centroids = mean(pairs, dim=3) # [B,N,K,3]
normals = cross_product(pairs) # [B,N,K,3]
normals = normals/norm(normals, dim=-1) # [B,N,K,3]
pos_mask = (normals[..., 0, 0]>0)*2-1 # [B,N,1,1]
normals = normals*pos_mask # [B,N,K,3]
normals = random_inverse(normals) # [B,N,K,3]
positions = sum(normals*centroids, dim=3)/sqrt(3)
      # [B,N,K,1]
features = concat([centroids, normals, positions],
    dim=2) # [B,N,K,7]
features = MLPs(features, out_channel=C) #[B,N,K,C]
features = pooling(features, dim=2) # [B,N,C]
out = concat([centroids, features], dim=2)
     # [B,N,3+C]
return out
```

Figure 1. Umbrella Repsurf Algorithm[3]

The number of parameters for PointNet++ Single Scale Point (SSG) is obtained as 1.476.791 which is slightly less than 1.83M indicated in the models table. The number of parameters for PointNet++ (SSG 2x) is obtained as 6.799.863 which is slightly less than 6.806 M indicated in the models table. We didn't use pretrained networks, though they are given, and retrained the network with default parameters to validate the results. 2

3.2 Datasets

The experiments reported in the paper consists of 3 different tasks, namely Classification, segmentation and object detection. However, source code for the object detection task is not included in the shared codebase which has reported to be conducted for ScanNet V2 and SUN-RGBD [7] datasets in the paper. Also, segmentation task is conducted for S3DIS [8] dataset, however it is not feasible for us to rerun experiments due to computational costs. Therefore, we have focused on classification task which is done with ModelNet40 [9] and ScanObjectNN [5] datasets. The codebase has only the ScanObjectNN implementation, so we used that dataset for full reproducibility and reported only for ScanObjectNN.

ScanObjectNN dataset consists of 11416 training and 2882 test samples. Each sample is a point cloud object with 2048 points including only the positional information. There is no additional info like color, intensity or surface normal in the original data. A link to the dataset is given by the authors and script for download process is also included in the authors' original codebase.

3.3 Hyperparameters

In appendix part of the original paper, subsection G, researchers mentioned about the hyperparameter setting for each downstream task. Here are the hyperparameters of the model.

- initial learning rate = 0.001
- total no of epochs = 250
- decay rate = 0.001
- optimizer = Adam
- Data augmentation = False
- 1024 sample points (n_points) sampled w/ farthest point sampling (FPS) from 2048 points
- Label smoothing with a ratio of 0.1.
- Batch size = 64.
- Seed = 2800
- Scheduler = Step scheduler is used.

Given the computational demands of the setup (with each training iteration taking approximately 10 hours), it was not feasible to perform a hyperparameter search. As a result, no optimization of hyperparameters was carried out.

3.4 Experimental setup and code

We have conducted experiments based on original implementations with previously given hyperparameters. We have modified environment variables and GPU selection in scripts to fit into our setup. Our setup is given in the following section.

3.5 Computational requirements

We run all our experiments in a setup with a RTX TITAN GPU with 24G VRAM. Average GPU time of each of our experiments, classification on ScanObjectNN dataset task, is around 12 hours per run. Authors have reported that they have run on a cluter of 4 x RTX 3090 GPUs and designed their codes for multi-GPU training. We have modified scripts for our single GPU.

4 Results

We were able to reproduce the experiments for proposed Umbrella RepSurf algorithm and achieved similiar results to the original paper. Still, there are minor differences in accuracy and model complexity (parameter count) which we believe is in a acceptable margin. Overall, we have conducted three experiments, two of them being PointNet++ w/Umbrella Repsurf (Single Scale Grouping) and one being PointNet++ w/Umbrella Repsurf (Multi-Scale Inference).

	PointNet++ w/ Umbrella (Single Scale Grouping)	PointNet++ w/ Umbrella (Multi-Scale Inference)	Best Score
Experiment 1	√		**84.84**
Experiment 2	√		**84.49**
Reported	√		**84.87**
Experiment 3		√	**85.70**
Reported		√	**86.05**

Table 1. Experimental vs Reported Results for Classification on ScanObjectNN.

	PointNet++ w/ Umbrella (Single Scale Grouping)	PointNet++ w/ Umbrella (Multi-Scale Inference)
Umbrella Surface Reconstructor	360	360
Surface Abstraction 1	14.272	53.120
Surface Abstraction 2	69.504	270.080
Surface Abstraction 3	731.136	1.064.448
Surface Abstraction 4	0	4.226.048
Classifier	661.519	1.185.807
Total	**1.476.791**	**6.799.863**
Reported	**1.483 M**	**6.806 M**

Table 2. Parameters of Experimental Models.

4.1 Results reproducing original paper

Explicitly Learning Local Information. Authors claim that, they do represent local geometry using second-order derivatives and they have generated approximate surface normals to implement this idea. According to our experiments, their implementation surpasses baseline which supports their claims.

Performance Review We have obtained the following accuracies from the reimplementation. The authors claims are indicated in reported row of the Table 1. As displayed in the table, we have achieved 85.70% on Multi-scale Inference (MSI) setup while they reported %86.05 accuracy.Our reproduced results are within 0.5% their claims which is a tolerable error rate for the experiments. Following the implementation of authors, reported scores calculated by running inference on test set every epoch and saving the top result for final score.

Computational Complexity Review We have observed during our experiments that the number of parameters are very close, though not exactly equal to the given values, to the numbers given in the paper. In the Table 2, we show the number of parameters layer by layer. The difference between multi-scale grouping (MSG) and single-scale grouping (SSG) comes from, first the additional fourth set abstraction layer and secondly increased the weights for each set abstraction layer by the factor 2.

5 Discussion

Overall, in this work we tried to reproduce a few of the experiments Ran et. al.[3] has conducted in the RepSurf original paper. We have followed publicly available original implementation and achieved the following conclusions. First of all, we were able to reproduce similar results to the the original ones, but the results are still not exactly the same. We believe that, this is due to the undeterministic behaviour of multi-threading and parallelisation in implementation details of PyTorch [10] which was not controllable by the authors. Yet still, we found the shared code base functionally complicated and structurally complex. We believe there are unnecessary function and module calls

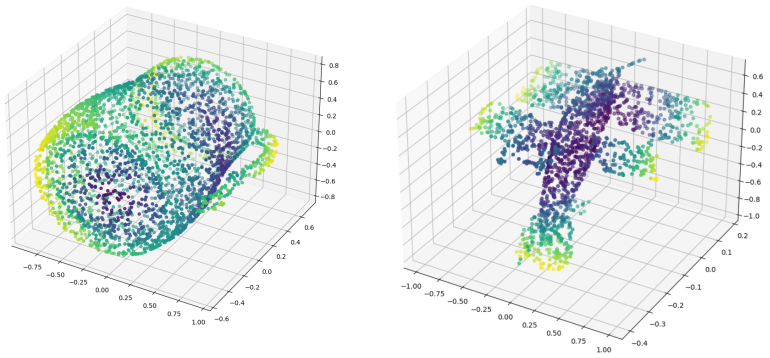

Figure 2. Example Cup and Plane Point Clouds

which makes code hard to track.
Secondly, we find the inference method of authors unethical and deceptive. They have followed a method that produces test results during the training, testing the data in each epoch and comparing with the best result, and if it greater, changing and reporting the best result. Optimizing the training process based on test data is not conventional and we find this paper morally not acceptable for scientific research.
Lastly, the shared codebase misses most of experiments and that weakens the reproducibility of other experiments such as Triangular RepSurf classification or the segmentation task. Due to our concerns from inference method, mentioned previously, we are highly sceptical about the other reported results.

5.1 What was easy

The original code implementation was easily accessible to the public and well-documented, with intuitively named variables and functions that reflected their roles in the project. The research paper also provided a comprehensive appendix with further implementation details and pseudo-codes. It is easy to start with the experiments, because the implementation included configuration files, download scripts for necessary datasets and pretrained weights.

5.2 What was difficult

Firstly, even though code was well written in terms of variable naming, the connections between the function and the modules of the code was problematic and hard to grasp. Also, the dataset is provided in a processed form in which labels are encoded without their true object name. Also in the paper, in the appendix part they used *random dropout* keyword which is a tautology. Due to its nature, dropout is already random. Lastly, authors have written their code in a way to run in their multi-GPU cluster without documenting the modifications to the environment variables. Therefore, we had difficulty to run the code in the first step with our environment.

5.3 Communication with original authors

We have not gotten in contact with original authors up to now.

References

1. C. R. Qi, H. Su, K. Mo, and L. J. Guibas. **PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation**. 2016.

2. C. R. Qi, L. Yi, H. Su, and L. J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space." In: **CoRR** abs/1706.02413 (2017). arXiv: 1706.02413.
3. H. Ran, J. Liu, and C. Wang. **Surface Representation for Point Clouds**. 2022.
4. H. Ran, J. Liu, and C. Wang. **RepSurf - Surface Representation for Point Clouds**. URL: https://github.com/hancyran/RepSurf.
5. M. A. Uy, Q.-H. Pham, B.-S. Hua, D. T. Nguyen, and S.-K. Yeung. **Revisiting Point Cloud Classification: A New Benchmark Dataset and Classification Model on Real-World Data**. 2019.
6. Y. Liu, B. Fan, S. Xiang, and C. Pan. **Relation-Shape Convolutional Neural Network for Point Cloud Analysis**. 2019.
7. S. Song, S. P. Lichtenberg, and J. Xiao. "SUN RGB-D: A RGB-D scene understanding benchmark suite." In: **2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. 2015, pp. 567–576.
8. I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. "3D Semantic Parsing of Large-Scale Indoor Spaces." In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. June 2016.
9. Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. **3D ShapeNets: A Deep Representation for Volumetric Shapes**. 2014.
10. A. Paszke et al. **PyTorch: An Imperative Style, High-Performance Deep Learning Library**. 2019.