A. Overview

This document provides technical details, additional quantitative results, and more qualitative test examples to the main paper. In Sect. B we provide derivations about the gradients back-propagated on three rotation angles and illustrate the construction of rotation matrices. In Sect. C we show more implementation details on our network architectures and training parameters. Then Sect. D illustrates comparison experiments between different optimization skills, while Sect. E shows more analysis experiments on our attack algorithm. At last, we show some visualization results in Sect. F.

B. Gradient Derivation and Rotation matrix Construction (Sect. 3.3)

Gradient Derivation. As illustrated in the main paper, rotating points along z axis by δ will increase the loss L by $\frac{\partial L}{\partial \phi_z}\delta$, where $\frac{\partial L}{\partial \phi_z}$ can be calculated by the chain rule as:

$$\frac{\partial L}{\partial \phi_z} = \sum_{i=1}^{n}\left(\frac{\partial x_i}{\partial \phi_z}\frac{\partial L}{\partial x_i} + \frac{\partial y_i}{\partial \phi_z}\frac{\partial L}{\partial y_i} + \frac{\partial z_i}{\partial \phi_z}\frac{\partial L}{\partial z_i}\right). \tag{1}$$

Here, ϕ_z refers to the rotation angle around z axis, which is the same as the azimuthal angle ϕ in the following spherical coordinate system (r, θ, ϕ):

$$\begin{cases} x = r\cos\phi\sin\theta, \\ y = r\sin\phi\sin\theta, \\ z = r\cos\theta. \end{cases} \tag{2}$$

Then, based on Eq. (2), we can write Eq. (1) as follows:

$$\begin{aligned} \frac{\partial L}{\partial \phi_z} &= \sum_{i=1}^{n}\left(\frac{\partial x_i}{\partial \phi}\frac{\partial L}{\partial x_i} + \frac{\partial y_i}{\partial \phi}\frac{\partial L}{\partial y_i} + \frac{\partial z_i}{\partial \phi}\frac{\partial L}{\partial z_i}\right) \\ &= \sum_{i=1}^{n}\left(-r\sin\phi\sin\theta * \frac{\partial L}{\partial x_i} + r\cos\phi\sin\theta * \frac{\partial L}{\partial y_i}\right) \\ &= \sum_{i=1}^{n}\left(-y_i\frac{\partial L}{\partial x_i} + x_i\frac{\partial L}{\partial y_i}\right). \end{aligned} \tag{3}$$

Similarly, for the remaining rotation axes ϕ_x and ϕ_y, we can calculate the gradients simply by rolling the coordinate system in Eq. (3) as follows:

$$\begin{aligned} \frac{\partial L}{\partial \phi_x} &= \sum_{i=1}^{n}\left(-z_i\frac{\partial L}{\partial y_i} + y_i\frac{\partial L}{\partial z_i}\right), \\ \frac{\partial L}{\partial \phi_y} &= \sum_{i=1}^{n}\left(-x_i\frac{\partial L}{\partial z_i} + z_i\frac{\partial L}{\partial x_i}\right). \end{aligned} \tag{4}$$

Rotation Matrix Construction. Given the optimized rotation angle $\Phi = [\phi_x, \phi_y, \phi_z]$, we construct the corresponding rotation matrices as follows:

$$R_{\phi_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_x & -\sin\phi_x \\ 0 & \sin\phi_x & \cos\phi_x \end{bmatrix}, \tag{5}$$

$$R_{\phi_y} = \begin{bmatrix} \cos\phi_y & 0 & \sin\phi_y \\ 0 & 1 & 0 \\ -\sin\phi_y & 0 & \cos\phi_y \end{bmatrix}, \tag{6}$$

$$R_{\phi_z} = \begin{bmatrix} \cos\phi_z & -\sin\phi_z & 0 \\ \sin\phi_z & \cos\phi_z & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{7}$$

Based on above equations, we compute the final rotation matrix $R = R_{\phi_z} \cdot R_{\phi_y} \cdot R_{\phi_x}$, where "$\cdot$" refers to the matrix multiplication.

C. Implementation Details

We implement ART-Point using PyTorch [1]. In detail, during attack, we set the step size of angle gradient descent $\alpha = 0.01$, a batch size $B = 17$ and adopt ten steps descent to obtain the adversarial rotation. During defense, we mainly use SGD to train existing point cloud classifiers following the same optimizer and learning rate schedules as used in their papers. We experiment with two optimization methods: iterative optimization and one-step optimization.

For the iterative optimization, we alternate the min-max process until the model converges. Specifically, to train a robust PointNet, in each iteration we use 10 epochs gradient descent on angles for maximization to find the most aggressive rotation angles and 50 epochs for minimization to train on adversarial datasets. We perform 10 iterations in total to obtain the final robust model.

For the one-step optimization, we construct the rotation pool by attacking multiple classifiers and reach the robust model in a single min-max iteration. Concretely, suppose that our target model is the PointNet classifier [2]. We not only attack PointNet but attack more robust classifiers such as PointNet++ [3] and DGCNN [4] to construct the rotation pool. We use 10 epochs gradient descent for maximization to find adversarial samples and 200 epochs for minimization to train on adversarial samples.

D. Comparison of Different Optimizations

We compare the training progress of the naive iterative optimization with the proposed one-step optimization. The experiments are conducted under ModelNet40 [5] and resulting classifiers are tested under randomly rotated datasets for evaluating the rotation robustness. We record the performance of three classifiers in each iteration and compare the final results with classifiers trained via the one-step method.

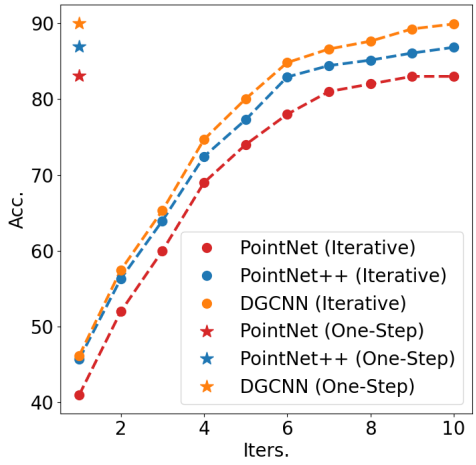

Figure 1. Adversarial training results of three classifiers under ModelNet40 [5] with different optimizations.

Descent Steps	$s = 9$	$s = 10$	$s = 11$	$s = 12$
	83.9	**84.3**	84.3	84.2
Rotation Angles	$[-\frac{1}{4}\pi, \frac{1}{4}\pi]$	$[-\frac{1}{2}\pi, \frac{1}{2}\pi]$	$[-\frac{3}{4}\pi, \frac{3}{4}\pi]$	$[-\pi, \pi]$
	87.2	86.4	85.5	84.3

Table 1. Adversarial training results under different settings.

Specifically, we follow the detailed implementations for both optimizations in Sect. C to reach robust models. It can be seen from Fig. (1) that for the iterative optimization it usually takes 8-10 iterations to reach the most robust model. In contrast, the one-step method obtains the robust model with competitive performance in one iteration. Note that, for different classifiers in one-step optimizations, the rotating pools are all constructed by attacking three models, *i.e.* PointNet [2], PointNet++ [3] and DGCNN [4].

E. More Ablation Studies

Here, we provide more control experiments to verify our rotation attack algorithm. We mainly conduct studies based on ModelNet40 [5] with PointNet classifiers [2].

Attack Step Size. We further illustrate experiments to select the appropriate step size in angle attacks. The results are shown in Tab. (2), where we record the average loss value of attacked samples under different step size α (rad). Our attack algorithm finds the most aggressive attacked samples that induce the highest loss with $\alpha = 0.01$.

Descent Steps and Rotation Angles. Finally, we verify the effect of different hyper-parameters on adversarial training. We adopt different descent steps during attacking and we also study the performance of our method under limited rotation ranges. The final results are shown in Tab. (1).

The adversarial training results tend to be saturated when the gradient descent step is large than 10, so we set the attack algorithm with 10 steps descent by defaults. Our method obtains better results under smaller rotation ranges, which demonstrates that by specifying the range of rotation angles, ART-Point can further increase the model robustness.

$\alpha = 0.1$	$\alpha = 0.08$	$\alpha = 0.06$	$\alpha = 0.04$	$\alpha = 0.02$
5.3	7.4	9.5	8.9	11.3
$\alpha = 0.01$	$\alpha = 0.008$	$\alpha = 0.006$	$\alpha = 0.004$	$\alpha = 0.002$
13.5	12.4	11.7	10.2	9.5

Table 2. Averaged loss values of attacked samples produced by attacks with different step sizes.

F. Visualization

Finally, we compare the classification loss of different models under the the randomly rotated test set of ModelNet40 [5] (Fig. 2) and ShapeNet16 [6] (Fig. 3). We illustrate the corresponding loss value under each rotated sample and compare them between the original DGCNN [4] and our best model ART-DGCNN. As can be seen, our method generally shows lower classification loss under both randomly rotated datasets.

References

[1] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 1

[2] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 1, 2

[3] 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. 1, 2

[4] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics*, 38(5):1–12, 2019. 1, 2, 3, 4

[5] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 1, 2, 3

[6] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (ToG)*, 35(6):1–12, 2016. 2, 4

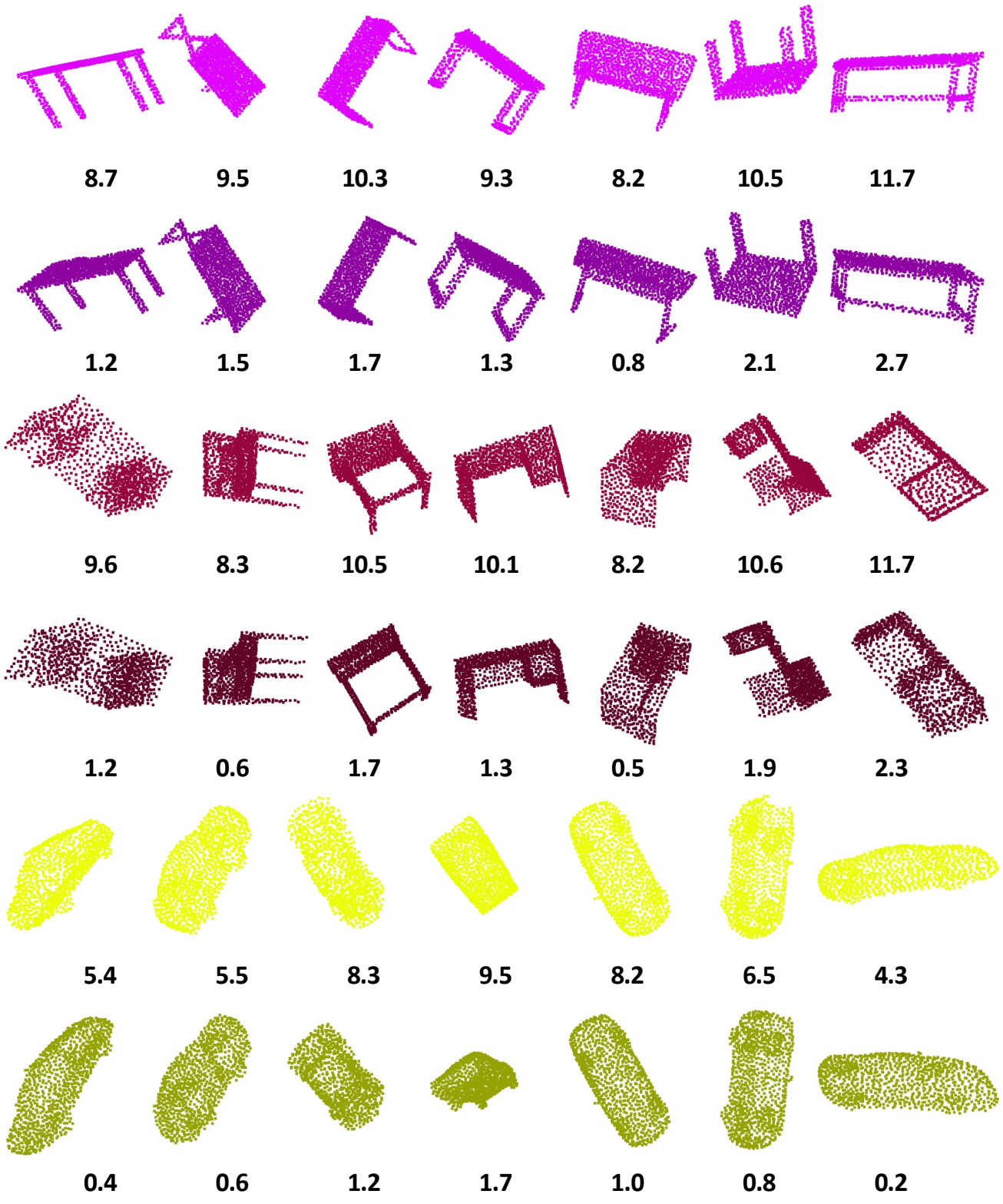

Figure 2. In every two rows, we compare the classification loss of DGCNN [4] (top row) and ART-DGCNN (bottom row) on the same arbitrarily rotated point clouds, which are randomly sampled from test sets of ModelNet40 [5]. From top to bottom, the categories of point clouds are "table", "desk" and "car".

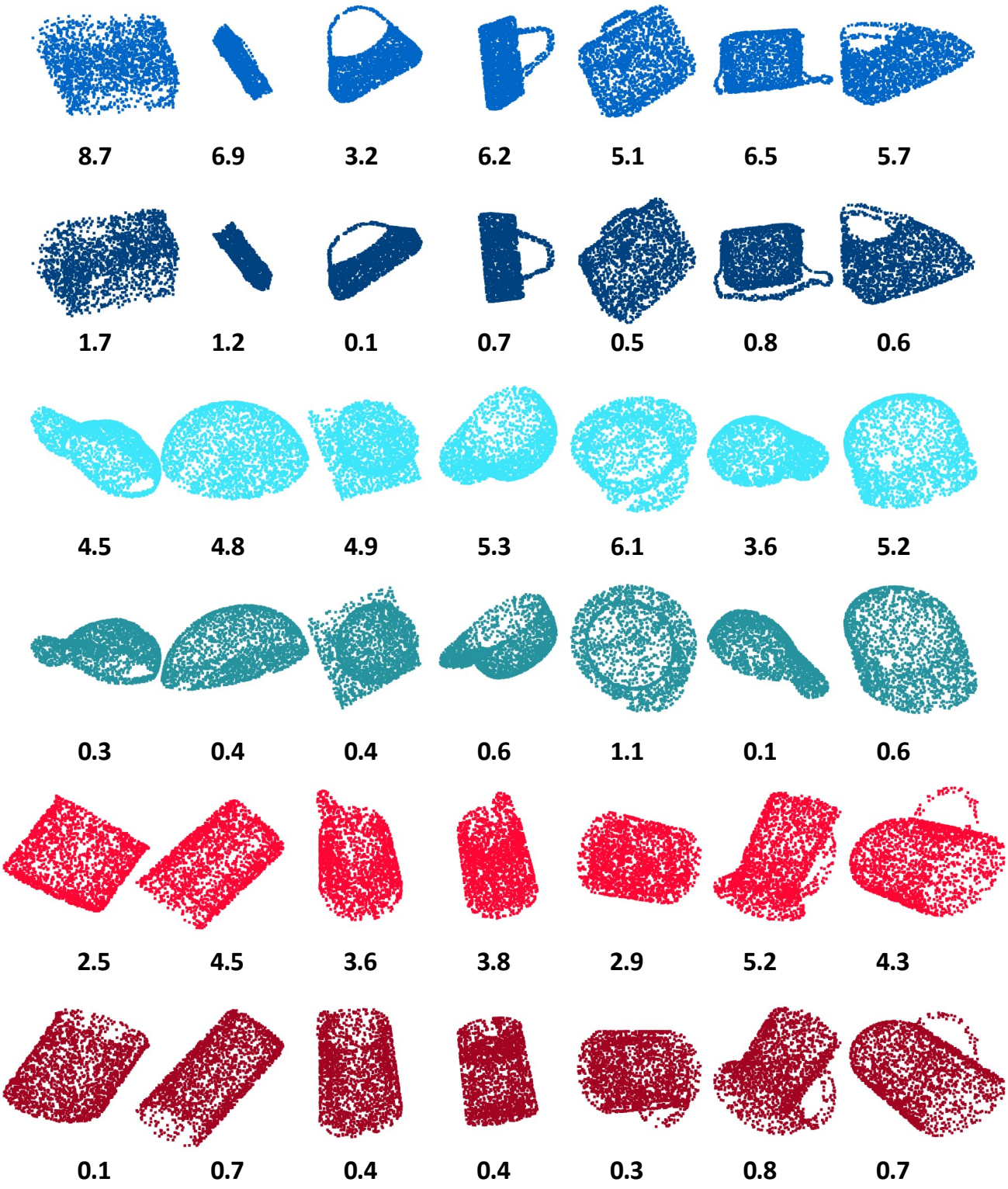

Figure 3. In every two rows, we compare the classification loss of DGCNN [4] (top row) and ART-DGCNN (bottom row) on the same arbitrarily rotated point clouds, which are randomly sampled from test sets of ShapeNet16 [6]. From top to bottom, the categories of point clouds are “bag”, “cap” and “mug”.