Applying 2D Adversarial Attacks on 3D Point Clouds

Matina Mahdizadeh Sani, Nima Jamali

April 28, 2024

1 Introduction

Vulnerabilities of deep neural networks to adversarial attacks and their increasing importance in deep learning have led to a greater need for robust models and adversarial defense mechanisms. Adversarial attacks create new images by making changes to the original image that are very close to the original one but are not correctly classified by the deep neural network. Two-dimensional attacks often involve perturbing the image and manipulating pixels. However, attacks on point clouds can take various forms. In these attacks, existing points are sometimes shifted to deceive the network. But the nature of point cloud sets, allows for other types of attacks as well. In some of these attacks, new points are generated and added to the existing set of points, while in others, some points are removed from the original set.

Contrary to adversarial attack techniques, adversarial defense methods are formulated to increase the resilience of deep neural networks against adversarial samples. Specifically, defensive methods aim to mitigate the decrease in classification accuracy of adversarial samples in comparison with the original ones.

The growing applications of 3D classification models in everyday scenarios have led to a greater significance of examining adversarial attack and defense methods than ever before. For example, in the domain of self-driving cars, inaccuracies in classifying objects detected by the vehicle's sensors can yield catastrophic outcomes. Therefore, if the deep neural network is not robust against perturbations in images or adversarial examples intentionally generated to deceive it, significant and sometimes irreversible damages may occur.

Adversarial attack methods identify vulnerabilities in neural networks, while existing defense methods aim to mitigate these vulnerabilities. Therefore, examining these attacks can lead to advancements in defensive techniques and increase the resilience of neural networks against adversarial samples.

While there have been numerous advancements in both attack and defense methods in the domain of 2D computer vision, less attention has been paid to three-dimensional data such as point clouds. Consequently, in this project, we investigated adversarial attack algorithms and extended some existing 2D attacks, including FGM [1], IFGM [2], PGD [3], and DeepFool [4], to the realm of 3D point clouds. We evaluated the performance of these attacks against two vanilla classification networks, namely PointNet [5] and DGCNN [6]. Additionally, we tested our attack against two simple defense

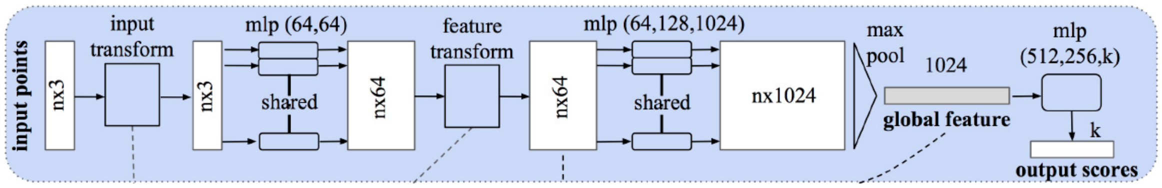

Figure 1: PointNet architecture [5].

mechanisms, SRS [8] and SOR [9]. Our results demonstrate that these attacks can also be effective in deceiving 3D classification networks.

2 Preliminaries

2.1 Point Clouds

Point clouds serve as a data structure used for representing multidimensional data. A three-dimensional point cloud comprises a set of points sampled from an object's surface. Each point is represented by coordinates along the x, y, and z axes. Furthermore, it can capture additional information like point color, intensity values, and more. Typically, point clouds are generated through three-dimensional laser scanners and LiDAR (Light Detection and Ranging) technology.

2.2 3D Classification Networks

2.2.1 PointNet

PointNet [5] is one of the first neural networks used for classification and segmentation of 3D point clouds. Its architecture is designed to adhere to two key properties of point clouds: being unordered, and invariant under transformations. To ensure that the output of the neural network remains unchanged under various transformations such as rotation, translation, etc., it employs a smaller network called T-net, which is similar to the main network. The output points of the T-net are then fed into a Multi-layer Perceptron (MLP) and then another T-net transformation to extract local features for each point.

The extracted local features are then fed as input to another MLP, and by applying max pooling on the output of this network, global features of the input point cloud are computed. Max pooling is a symmetric function, and its output remains constant regardless of the order of points.Lastly, these global features are fed into another MLP, determining the score for each class. This network's architecture is visible in Figure 1.

2.2.2 DGCNN

Another neural network architecture considered for point cloud classification in this project is DGCNN (Dynamic Graph Convolutional Neural Network) [6]. This network aims to leverage Convolutional Neural Networks (CNNs) used for 2D image classifi-

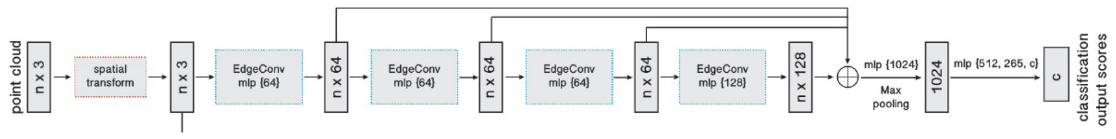

Figure 2: DGCNN architecture [6].

cation while also incorporating geometric relationships between points, in addition to local point-wise ones. To achieve this, it employs a structure called EdgeConv.

In EdgeConv, a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is first constructed, where $\mathcal{V}$ represents the set of graph vertices containing all points, and $\mathcal{E}$ represents the set of edges. In this graph, each point is connected to its k nearest neighbors, forming edges. Additionally, each vertex has a self-loop (an edge to itself). It's important to note that this structure is used in different layers of the neural network. Consequently, the nearest neighbors of a vertex may differ across these layers, resulting in a dynamic graph structure between points.

Also, it is worth mentioning that the EdgeConv structure remains invariant to different permutations of points and their translations. The architecture of this network is visible in Figure 2.

2.3 3D Adversarial Defense

SRS [8] (Simple Random Sampling) randomly chooses and eliminates n points from a point cloud's set of points. The primary drawback of this method lies in its random point removal, which may overlook points generated by adversarial attack methods and remove original ones.

Additionally, SOR [9] (Statistical Outlier Removal), aims to detect and eliminate outlier points. To achieve this, it calculates the average distance from each point to its k nearest neighbors. Subsequently, it computes the average $(\bar{d})$ and standard deviation (σ) of these distances. Points, such as x_i, for which $d_i \geq \bar{d} + \alpha.\sigma$, are then removed.

While there are numerous successful defense methods applicable to 3D adversarial point clouds, we chose to assess our generated adversarial samples using the aforementioned methods primarily because of their simplicity and low-resource consumption.

3 Adversarial Attack Methods

In this section, we applied adversarial attacks previously implemented in 2D to 3D point clouds using existing 3D architectures (such as PointNet and DGCNN). After applying these attacks and generating adversarial examples, we employed SRS and SOR defenses to preserve accuracy. In the following subsection different attacks that were extended to work on 3d point clouds are explained in details.

3.1 FGM

The FGM [1] (Fast Gradient Method) adversarial attack is a powerful technique employed in the context of adversarial machine learning. This method operates by perturbing input data samples in a manner that maximizes the loss function of a neural network model, causing misclassification. FGM computes the gradient of the model's loss function with respect to the input data and adjusts the input data in the direction that increases the loss the most. The adversarial data is computed as:

$$x^* = x + \epsilon[\nabla_x(C(\theta, x, y))], \tag{1}$$

where ϵ represents the magnitude of the perturbation, $\nabla_x(C(\theta, x, y))$ is the gradient of the loss function with respect to the input data x for given model parameters θ and true label y. The motivation behind the method is to increase the cost when classifying true label, so the model would predict another class with lower cost.

There is a signed version of this method in 2D, called FGSM [1], which uses the sign of gradient instead of the gradient itself. The code for this approach is also implemented in the project, but it is not reported since the result was not very promising in compare to the unsigned version.

Targeted FGM is a variant of the FGM adversarial attack where instead of simply maximizing the loss function to cause misclassification, the attacker aims to steer the model's prediction towards a specific target class. This means the adversarial perturbations are crafted to cause the model to classify the input as the specified target class rather than any other class. In this variation, the adversarial data is computed as:

$$x^* = x - \epsilon[\nabla_x(C(\theta, x, t))], \tag{2}$$

There are different techniques for choosing the target classes. In this project the targets are assigned randomly for simplification. Same as signed FGM, this version is also not reported in this file, since the results were not good. However, the code and results on these versions for all attacks can be found on the relevant notebook.

3.2 IFGM

IFGM [2] (Iterative Fast Gradient Method) is an extension of the FGM adversarial attack. While FGM generates adversarial examples by applying a single perturbation to the input data, IFGM iteratively applies small perturbations to gradually steer the input towards the desired misclassification. The IFGM algorithm typically involves repeating the perturbation process for a fixed number of iterations, which by experiments, we concluded that 10 iteration is enough for this project. At each iteration, the perturbation is computed based on the gradient of the model's loss function with respect to the input data, similarly to FGM. The adversarial data is computed as:

$$x^{(0)} = x \tag{3}$$

$$x^{(i)} = x^{(i-1)} + \alpha[\nabla_x(C(\theta, x, y))], \tag{4}$$

where α represents the magnitude of the perturbation in each iteration, and $\nabla_x(C(\theta, x, y))$ is computed as before.

Same as FGM, a signed and targeted version of this method are also implemented. The adversarial data for the targeted IFGM is calculated as:

$$x^{(0)} = x \tag{5}$$

$$x^{(i)} = x^{(i-1)} - \alpha[\nabla_x(C(\theta, x, t))], \tag{6}$$

3.3 PGD

PGD [3] (Projected Gradient Descent) is another variant of the iterative adversarial attack, closely related to the IFGM. PGD iteratively perturbs input data to create adversarial examples, but it introduces an additional step to ensure that the perturbed data remains in a specified range or constraint. The projection step is performed before applying any gradient updates, and project $x^(1)$ onto a constrained ball around the original input data. This ensures that the perturbed data starts within the specified perturbation budget. The adversarial data is computed as:

$$proj = uniform(\frac{-\epsilon}{dim(x)}, \frac{\epsilon}{dim(x)}) \tag{7}$$

$$x^{(0)} = x + proj \tag{8}$$

$$x^{(i)} = x^{(i-1)} - \alpha[\nabla_x(C(\theta, x, t))]. \tag{9}$$

where α and $\nabla_x(C(\theta, x, t))$ are similar to what we defined in IFGM method. Moreover, ϵ is the magnitude of the perturbation, $uniform$ represents a uniform distribution, and dim is dimension of x.

Our implementation also covers signed and targeted version of this attack, where the adversarial data for the targeted attack is computed as:

$$proj = uniform(\frac{-\epsilon}{dim(x)}, \frac{\epsilon}{dim(x)}) \tag{10}$$

$$x^{(0)} = x + proj \tag{11}$$

$$x^{(i)} = x^{(i-1)} - \alpha[\nabla_x(C(\theta, x, t))], \tag{12}$$

3.4 DeepFool

The key insight behind DeepFool [4] is that the decision boundaries of deep neural networks are often linear in high-dimensional feature spaces, making it relatively straightforward to compute adversarial perturbations using linear approximation methods. In this project, we extended this approach, originally proposed for 2D images, to the realm of 3D point clouds.

Based on the introduction provided, DeepFool is an untargeted attack algorithm that aims to misclassify a given input by introducing perturbations in a manner that shifts the generated adversarial sample outside of original sample's decision boundary. In multiclass linear classifiers, these decision boundaries form polyhedrons. Figure 3

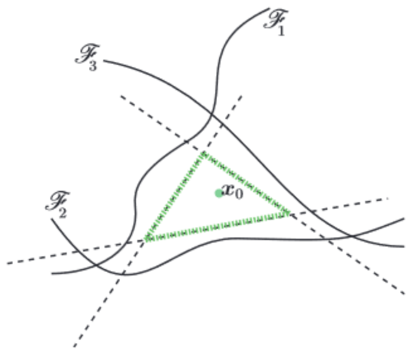

Figure 3: Illustration of DeepFool attack [4].

illustrates this concept, depicting the minimum perturbation required for misclassification, which is equivalent to the distance from the closest hyperplane. Consequently, at each iteration, the algorithm identifies the class index associated with the minimum perturbation using the following formula:

$$\hat{l}(x^{(i)}) = \underset{k \neq \hat{k}(x^{(i)})}{\arg\min} \frac{|f_k(x^{(i)}) - f_{\hat{k}(x^{(i)})}(x^{(i)})|}{\|\nabla f_k(x^{(i)}) - \nabla f_{\hat{k}(x^{(i)})}(x^{(i)})\|_2} \tag{13}$$

Upon determining the minimum perturbation l, the algorithm computes the perturbation r^* as follows:

$$r^*(x^{(i)}) = \frac{|f_k(x^{(i)}) - f_{\hat{k}(x^{(i)})}(x^{(i)})|}{\|\nabla f_k(x^{(i)}) - \nabla f_{\hat{k}(x^{(i)})}(x^{(i)})\|_2^2} [\nabla f_k(x^{(i)}) - \nabla f_{\hat{k}(x^{(i)})}(x^{(i)})] \tag{14}$$

DeepFool iteratively computes the direction in which the input data needs to be perturbed to cause misclassification. The algorithm updates the input data by adding the computed perturbation, incorporating an overshoot hyperparameter η to ensure reaching the other side of the classification boundary:

$$x^{(i+1)} = x^{(i)} + (1 + \eta) r^*(x^{(i)}) \tag{15}$$

This process is repeated until the model misclassifies the input or the maximum number of iterations is reached.

4 Experiments

This section investigates the experimental evaluation of the proposed method on ModelNet40 dataset and compares the results of different attacking algorithms against classification models and adversarial defenses.

4.1 Dataset

In this project, we used the ModelNet40 dataset, comprising three-dimensional point clouds from 40 distinct categories. This dataset contains a total of 12,311 distinct point

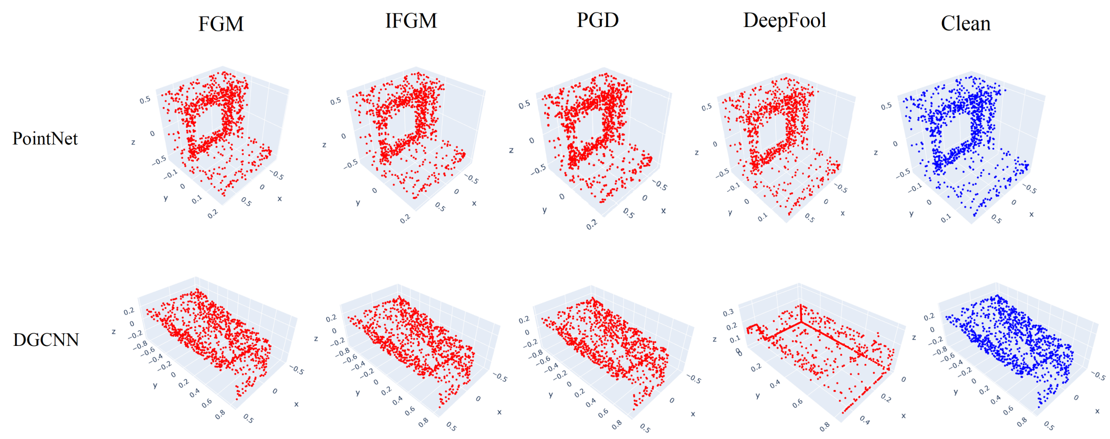

Figure 4: Illustration of original and adversarial point clouds.

clouds, out of which 9,843 are used for training and 2,468 for testing. In the experiments conducted in this project, we use a version of this dataset available in [7].

4.2 Experimental Setup

We applied all the introduced attacking algorithms to the entire test set, resulting in 2,468 adversarial examples per attack. As previously mentioned, for targeted attacks, we randomly assigned a target class to each point cloud and maintained this target across all attacks. The settings and hyperparameters for classification networks and defense methods were almost similar to those described in their respective papers. However, we trained PointNet [5] for 200 epochs and DGCNN [6] for 100 epochs, and employed the best-performing models based on test-set accuracy in our experiments.

For FGM, IFGM, and PGD, we set $\epsilon = 0.2$, $\alpha = 0.02$, and used 10 iterations in iterative methods. Additionally, we employed normalized gradients in all three methods to ensure that the generated adversarial examples closely resembled the original point clouds. In DeepFool, which is the only attack among the methods we investigated that has just an untargeted version, we limited the number of iterations to 20 while setting $\eta = 0.02$.

4.3 Results

In this section, we evaluate the results of the proposed attacking algorithms against 3D classification models, along with SRS [8] and SOR [9]. It is important to note that we will only present the results for untargeted attacks in this document. However, the results for targeted attacks can be found in the corresponding Jupyter Notebook files attached to this project.

The evaluation metric considered in this project is accuracy, which represents the percentage of samples classified correctly. Table 1 displays the results for PointNet, while Table 2 presents the results for DGCNN.

While we observe that all implemented attacks effectively decrease the classification

Table 1: Comparison of different attacking algorithms in terms of accuracy against PointNet and different defense methods.

	Attacks				
Defenses	Clean	FGM	IFGM	PGD	DeepFool
No defense	87.32%	41.90%	24.51%	24.19%	3.48%
SRS	NA	50.12%	43.48%	41.90%	6.16%
SOR	NA	58.68%	53.16%	52.67%	6.32%

Table 2: Comparison of different attacking algorithms in terms of accuracy against DGCNN and different defense methods.

	Attacks				
Defenses	Clean	FGM	IFGM	PGD	DeepFool
No defense	90.84%	77.96%	60.58%	60.74%	3.48%
SRS	NA	83.27%	83.14%	82.58%	25.61%
SOR	NA	82.01%	79.13%	78.85%	15.03%

accuracy, it's evident that DeepFool outperforms the other attacks, with PGD and IFGM performing similarly to each other (with PGD slightly outperforming IFGM). As expected, FGM is the least effective method among them since it does not compute the adversarial example iteratively. Additionally, upon comparing Table 1 and Table 2, we find that DGCNN generally shows greater robustness to adversarial samples compared to PointNet. Surprisingly, SRS performed better than SOR when combined with DGCNN.

Also, there are some qualitative results in Figure 4, depicting point clouds after applying different attacks. As you can see from the image, despite the good performance of the DeepFool attack in terms of quantitative results, it can generate some failure cases where point clouds are significantly altered after the attacks. In contrast, this observation is not made with other attacks, as the point clouds consistently resemble the original ones.

You can find all generated checkpoints and adversarial samples, as well as the source code for different modules in here.

5 Conclusion

In this project, we implemented and tested various adversarial attack algorithms originally designed for 2D image classification on 3D point clouds. We evaluated the performance of these extended attacks on the ModelNet40 dataset against original 3D classification networks and two adversarial defense methods. Our findings indicate that all of these methods can effectively deceive the networks to some degree, demonstrating their efficacy in the realm of 3D point clouds.

References

[1] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[2] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[3] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[4] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[5] 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.

[6] 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 (ToG)*, 38(5):1–12, 2019.

[7] Chong Xiang, Charles R Qi, and Bo Li. Generating 3d adversarial point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9136–9144, 2019.

[8] Jiancheng Yang, Qiang Zhang, Rongyao Fang, Bingbing Ni, Jinxian Liu, and Qi Tian. Adversarial attack and defense on point sets. *arXiv preprint arXiv:1902.10899*, 2019.

[9] Hang Zhou, Kejiang Chen, Weiming Zhang, Han Fang, Wenbo Zhou, and Nenghai Yu. Dup-net: Denoiser and upsampler network for 3d adversarial point clouds defense. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1961–1970, 2019.