Random Walks for Adversarial Meshes

Amir Belder
Technion – Israel Institute of Technology
Israel
amirbelder@campus.technion.ac.il

Gal Yefet
Technion – Israel Institute of Technology
Israel
galyefet@campus.technion.ac.il

Ran Ben Izhak
Technion – Israel Institute of Technology
Israel
benizhakran@gmail.com

Ayellet Tal
Technion – Israel Institute of Technology
Israel
ayellet@ee.technion.ac.il

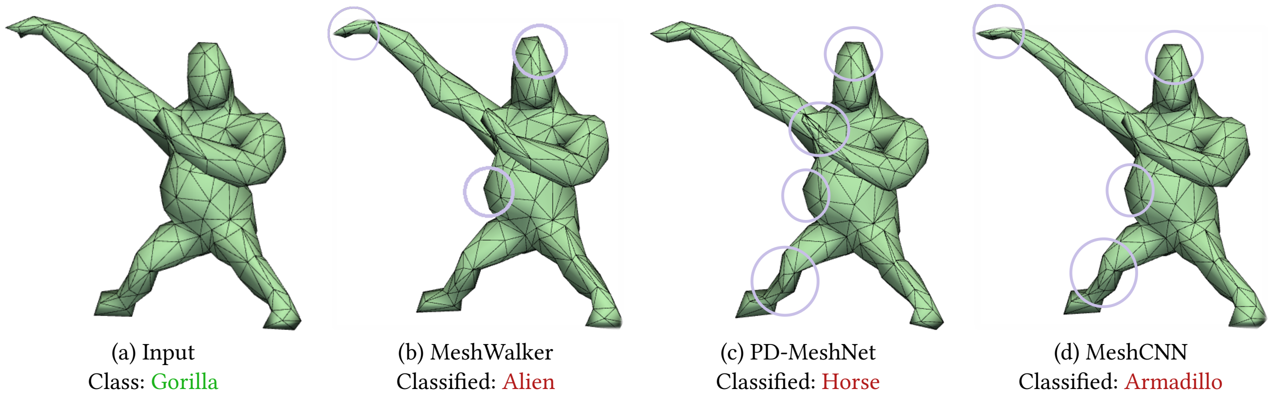


Figure 1: Adversarial meshes. (a) Given a mesh of a gorilla, it is attacked by modifying the mesh regions proportionally to the influence they have on the specific SOTA classification network [Hanocka et al. 2019; Lahav and Tal 2020; Milano et al. 2020]. The most modified regions are marked by gray circles. Although the modifications are barely visible, all the networks are misled and misclassify the gorilla as either an alien, a horse or an armadillo.

ABSTRACT

A polygonal mesh is the most-commonly used representation of surfaces in computer graphics. Therefore, it is not surprising that a number of mesh classification networks have recently been proposed. However, while adversarial attacks are wildly researched in 2D, the field of adversarial meshes is under explored. This paper proposes a novel, unified, and general adversarial attack, which leads to misclassification of several state-of-the-art mesh classification neural networks. Our attack approach is black-box, i.e. it has access only to the network's predictions, but not to the network's full architecture or gradients. The key idea is to train a network to imitate a given classification network. This is done by utilizing random walks along the mesh surface, which gather geometric information. These walks provide insight onto the regions of the mesh that are important for the correct prediction of the given classification network. These mesh regions are then modified more than other regions in order to attack the network in a manner that is barely visible to the naked eye.

KEYWORDS

mesh, neural networks, geometry, 3d adversarial attacks

ACM Reference Format:
Amir Belder, Gal Yefet, Ran Ben Izhak, and Ayellet Tal. 2022. Random Walks for Adversarial Meshes. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings (SIGGRAPH '22 Conference Proceedings), August 7–11, 2022, Vancouver, BC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3528233.3530710

1 INTRODUCTION

Neural networks achieve outstanding results in numerous tasks in computer vision & graphics, however they are oftentimes vulnerable to adversarial attacks [Andriushchenko and Flammarion 2020; Biggio et al. 2013; Carlini and Wagner 2017, 2018; Croce and Hein 2019; Ebrahimi et al. 2018; Madry et al. 2018; Papernot et al. 2017; Rony et al. 2019; Szegedy et al. 2014; Tsai et al. 2020; Wallace et al. 2019; Xie et al. 2019]. These attacks modify the input data in a way that is hardly visible to the naked eye, yet leads to misclassification. Adversarial attacks can be divided into categories: *white-box*, *black-box*, and *gray-box*. In white-box, the attacker has access to the

SIGGRAPH '22 Conference Proceedings, August 7–11, 2022, Vancouver, BC, Canada
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9337-9/22/08…$15.00
https://doi.org/10.1145/3528233.3530710

networks' full architecture, gradients and predictions [Goodfellow et al. 2015; Papernot et al. 2016]. In black-box, the attacker has access only to the networks' predictions [Papernot et al. 2016]. In gray-box, the attacker has access to more than just the predictions, but not to the full architecture [Vivek et al. 2018].

This paper focuses on three-dimensional data, differently from the works mentioned above that attack images. In 3D, most works attempt to attack point-cloud networks [Hamdi et al. 2020; Liu et al. 2019; Wicker and Kwiatkowska 2019; Xiang et al. 2019; Zhang et al. 2021; Zhao et al. 2020]. The key idea is to move specific points, whose displacement will lead to misclassification. Since point clouds have no topological constraints, such movements would be hardly visible. Polygonal meshes, however, are inherently different in that aspect. Moving even a single vertex might cause a highly noticeable movement of the adjacent edges & faces and possibly result in topological changes such as self-intersections. If the meshes are textured the attacks may change the texture of the meshes to cause misclassification [Xiao et al. 2019; Yang et al. 2018; Yao et al. 2020]. There are also works that attack graph networks [Chen et al. 2017; Sun et al. 2018; Zhang et al. 2019]. These attacks modify the graph structure, which is undesirable for meshes.

We address the task of attacking the most popular representation of 3D data—texture-less meshes. As this representation is used in many applications, including modeling, animation and medical purposes, acknowledging the vulnerabilities of mesh classification networks is important. For meshes, white-box attacks were proposed [Mariani et al. 2020; Rampini et al. 2021; Zhang et al. 2021].

We propose a novel, black-box, unified, and general adversarial attack, which leads to misclassification in SOTA mesh neural networks, as shown in Fig 1. At the base of our method lies the concept of an *imitating network*. We propose to train a network to *imitate* a given classification network. For each network we wish to attack, our imitating network gets as input pairs of a mesh & a prediction vector for that mesh (i.e. querying all meshes in the dataset). It basically learns the classification function of the given attacked network, by learning to generate prediction vectors for the meshes. For this to be done, our loss function should consider the distribution of the prediction vectors rather than one-hot label vectors used for classification.

Our imitating network utilizes random walks along the mesh. As shown in [Lahav and Tal 2020], random walks are a powerful tool for mesh exploration, gathering both global and local geometric information on the object. During a walk, features of various importance are extracted, as obviously some regions of the mesh are more distinctive than others. The state-of-the-art classification networks inherently differ from each other, both in the manner they extract their features and in the parts of the meshes that they focus on [Feng et al. 2019; Hanocka et al. 2019; Lahav and Tal 2020; Milano et al. 2020]. Therefore, architecture-dependent changes are needed in order to cause each of the SOTA systems to misclassify. We will show that our random walk-based network manages to learn these specific architecture-dependent features.

Our objective, however, is not only to cause misclassification, but also to do so while minimizing the change to the input meshes. If our modifications focus solely on the distinct regions, the network is going to be misled indeed, but the attacked meshes will look bad. For instance, a camel with no hump is not a distinctive camel. Thus, the two goals of the attack—misclassification and remaining distinct—might contradict. We will show that random walks suit these contradicting goals: As they wander around the surface, the modifications are spread across the entire mesh, both in distinctive and in non-distinctive regions. The need to spread the modifications is enhanced by [Ben Izhak et al. 2022]'s observation that all vertices contribute to the classification, though to varying degrees. In our method, as a vertex's influence on classification grows, so does its modification. This influence is manifested in the gradient of the classification loss. Therefore, we change the mesh vertices in the opposite direction of the gradients.

Our approach is shown to fool SOTA classification networks on two of the most commonly-used datasets. For each dataset, our approach attacks the networks that report the best results for that dataset. In particular, we attack (1) MeshCNN [Hanocka et al. 2019], PD-MeshNet [Milano et al. 2020] and MeshWalker [Lahav and Tal 2020] on SHREC11 [Veltkamp et al. 2011] and (2) MeshNet [Feng et al. 2019] and MeshWalker [Lahav and Tal 2020] on ModelNet40 [Wu et al. 2015]. These datasets are chosen not only due to their prevalence, but also because the SOTA networks achieve excellent results for them, which is important in order to verify that excellent classifiers can be misled. As a typical example, PD-MeshNet achieves 99.7% accuracy before the attack and 18.3% after it on SHREC11.

Fig. 1 illustrates how very small modifications to the original mesh, which would seem insignificant to a person, cause different SOTA systems to misclassify the object. Moreover, the figure shows how different parts of the mesh are changed in order to mislead each of the networks: The right hand of the Gorilla is modified in order to fool MeshWalker, the left hand and the lower leg to mislead PD-MeshNet, and the right hand and the lower leg to fool MeshCNN; interestingly, the head and the belly are modified in all cases. This is an additional benefit of our network—it detects the parts of the object each of these networks focuses on for classification. This provides a rare opportunity to shed some light on how these networks classify.

This paper makes the following contributions: First, it presents a novel, unified and general black-box approach to attack mesh classification networks. Furthermore, it proposes a network that realizes this approach. Second, it demonstrates how vulnerable to adversarial attacks the current SOTA classification networks are. This can be useful for developing more robust networks. Moreover, profound insight is gained onto the parts of the mesh that are important for the correct prediction of the different networks.

2 METHOD

We are given a dataset of meshes, partitioned into categories, and all the prediction vectors of the classification network to be attacked. Adversarial attacks aim to change the input in a manner that would cause the network to misclassify the input, while a person would still classify it as its true class. In the following we propose an approach that, when given a mesh, will generate such a modified mesh.

The basic question is where on the mesh these changes shall be and how they shall be spread. Evidently, distinctive regions must be modified in order to cause misclassification. However,

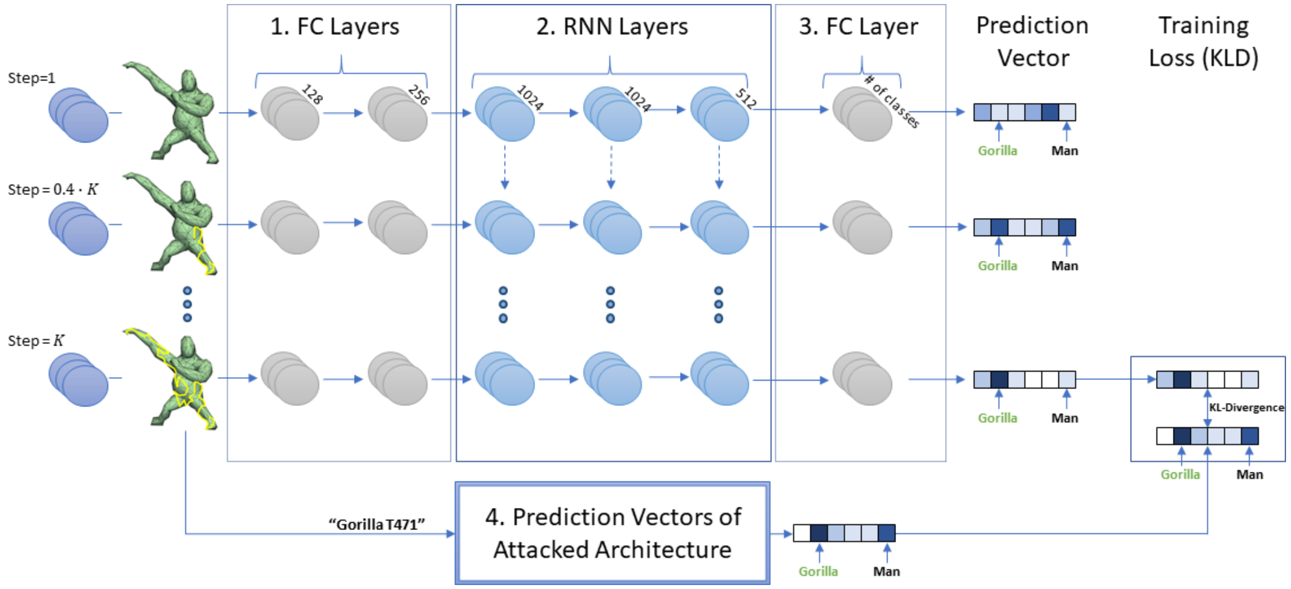


Figure 2: Imitating network architecture. Our network gets a mesh, a random walk (the yellow walk on the gorilla) and a prediction vector. As in [Lahav and Tal 2020], it consists of 3 components: (1) The FC layers change the feature space; (2) The RNN layers aggregate the information along the walk; (3) The FC layer predicts the outcome of the network. The KLD loss is applied to the prediction of the last walk vertex jointly with the imitated network prediction.

if the modifications are concentrated on these regions, they are likely to be noticeable to the naked eye, which will not satisfy our constraint. This creates a delicate equilibrium that must be upheld for a successful adversarial attack: spreading the change across the mesh, while still fooling the classification network.

Our key idea is to randomly walk along the mesh edges and slightly move the vertices along the walk. A *random walk* is extracted by randomly choosing an initial vertex, and then iteratively adding vertices to the sequence from the adjacent vertices to the last vertex of the sequence (if it is not already there). We draw inspiration from the MeshWalker network [Lahav and Tal 2020], which shows that random walks are a powerful tool for classification. At the base of our method lies the observation that some portions of each walk influence the classification more than others. Furthermore, the amount of influence is manifested in the gradient of the classification loss. The more influential a vertex is , the larger its gradient will be. Thus, our attack is gradient-based: By changing the vertices against the gradients, the most influential vertices will change the most, thereby changing the mesh in a manner that makes it less likely to be classified correctly. However, as all the vertices of the walk contribute to the classification, all of them will change during the attack, hence spreading the changes also to non-distinctive areas. This is crucial in achieving the above target equilibrium.

Our attack approach is general and unified, i.e. it can be applied to any mesh classification network. In particular, for a given classification network we wish to attack, we train an *imitating network*. Given meshes and their corresponding prediction vectors, queried from the network it shall attack, the imitating network learns similar attributes to those of the imitated network. We elaborate below.

2.1 Training

Each imitating network gets as input: (1) the train meshes of a dataset and (2) the prediction vectors of the attacked network on these meshes, i.e. the full probability vectors of the dataset classes. The goal is to learn to imitate the prediction function, so as to learn the same traits as the network it is imitating. We note that each classification network preforms a different type of non-trainable pre-processing to the meshes before classifying them, which usually includes mesh simplification. We preform the same pre-processing.

Regardless of the network we wish to attack, our imitating network has the architecture illustrated in Fig. 2. It is similar to the architecture of MeshWalker [Lahav and Tal 2020], except for two differences: the input and the loss function. Given a random walk that consists of vertices (3D coordinates) along the mesh, referred to as *steps*, the data is aggregated as follows. First, a couple of FC layers upscale each vertex into 256 dimensions. The second component is a *recurrent neural network (RNN)*, whose defining property is being able to "remember" and accumulate knowledge. Thus, it aggregates the information of the vertices, "remembering" the walk's history. A final FC layer predicts the classification. The objective of our imitating network differs from that of [Lahav and Tal 2020]: Rather than predicting the classes of the meshes, our imitating network drives to imitate a specific function—the prediction function of the imitated network. Therefore, rather than requiring one-hot vectors of the source classes of the meshes as input, the imitating network requires the full probabilities vectors of the system it is imitating. This is reflected in the type of the loss function used for training.

We use *Kullback–Leibler divergence (KLD)* loss function, instead of the sparse cross-entropy of [Lahav and Tal 2020], as we aim measure a distance between probabilities. Let P_θ be the posterior

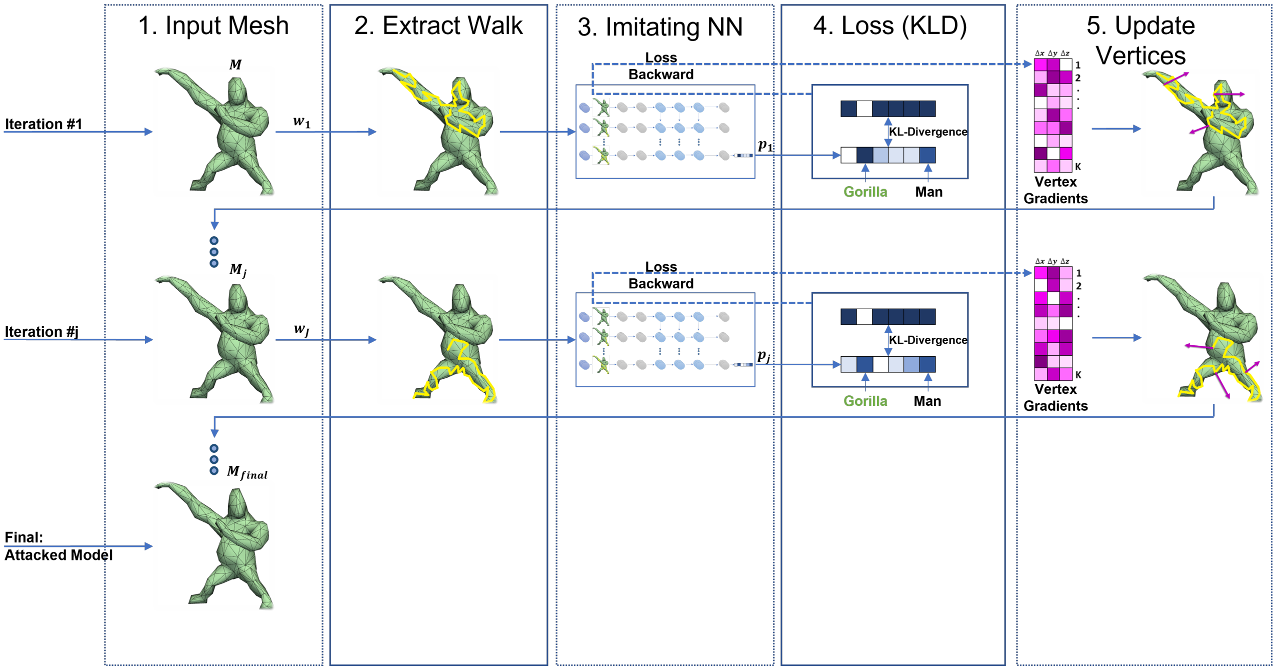


Figure 3: Attack. At every iteration, a walk is extracted and our imitating network is applied to this walk, producing a prediction vector. The gradients are computed for each vertex along the walk and the vertex location is updated w.r.t its gradient. The modified mesh is fed into the next iteration, with a new walk.

probability for observation O, computed by the trained network, parameterized with θ, P_{ref} be the given ground truth probability for O, t_i (/ r_i) be the probability that the input mesh belongs to class i out of D classes in the dataset. The loss function is defined as:

$$KLD(P_\theta(t),\ P_{\text{ref}}(r)) = \sum_{i=1}^{D} p_{\text{ref}}(r_i) \log \frac{p_{\text{ref}}(r_i)}{p_\theta(t_i)}. \tag{1}$$

2.2 Attack.

Once an imitating network $\mathcal{N}$ of a given mesh classification network is trained, the latter can be attacked by $\mathcal{N}$, as described in Algorithm 1. Given a mesh $\mathcal{M}$ to be attacked, the following process iterates until $\mathcal{N}$ predicts the class of $\mathcal{M}$ as something other than its given class: First, a random walk w_j is extracted and $\mathcal{N}$ produces its prediction for w_j, p_j. If the predicted class differs from $\mathcal{M}$'s true class, $\mathcal{M}$ is saved as the attacked model, i.e. this is the stopping condition (in practice, we exit when misclassification occurs a few times). Otherwise, the *KLD* Loss between the prediction and the given one-hot label vector is calculated. The gradients of the walk's vertices are calculated according to the result of the loss, as explained below. Finally, the walk vertices are changed in the opposite direction of their gradients. In effect, the vertices of the walk that contribute more than others to the classification will change more. Every additional walk modifies the previous slightly-modified mesh. Fig. 3 illustrates our attack.

Specifically, our objective is to update the input mesh according to the network's loss function and prediction. This is somewhat similar to the objective of back-propagation during training, where the weights of the network are updated according to the effect they had over the network's loss function and prediction. In both cases, the influence is manifested in the derivative of the loss function. Thus, we wish to measure how each vertex along the walk influences the loss and to change the vertex's coordinates accordingly, in order to cause misclassification. Recall that our trained imitating network is *RNN*-based. Thus, it aggregates the data of the walk, vertex by vertex, and "remembers" the history of the walk. The network's memory of each step along the walk is manifested in its *state*, which is updated after every step. (The last state is also used for prediction.) Neural networks compute the gradients backwards for every layer, all the way, up to the input layer of the model (the trainable parameters in adversarial attacks). Therefore, we get the

ALGORITHM 1: Adversarial Attacks via Random Walks

```
Input: A mesh $\mathcal{M}$, a one-hot label vector (of $\mathcal{M}$) $r$, an imitating
       network $\mathcal{N}$
Output: Attacked model $\mathcal{M}(final)$
for $j \leftarrow 1$ to $MAX\_ITERATIONS$ do
    $w_j \leftarrow ExtractWalk(M)$;
    $p_j \leftarrow Prediction(N, w_j)$;
    if $Softmax(p_j)! = r$ : then
        $Return\ M$ (final);
    end
    $loss \leftarrow KLD(p_j, r)$;
    $gradients \leftarrow CalcGradients(loss, w_j)$;
    $M \leftarrow UpdateMesh(M, gradients)$;
end
```

gradient for each vertex, which is a 3D vector, i.e. the size of the input we started with.

In the attack's last stage, we modify the coordinates of a vertex according to the above gradient, by moving the vertex by $(\Delta x, \Delta y, \Delta z)$, that is computed by multiplying the gradient by a small constant, α: $(\Delta x, \Delta y, \Delta z) = \alpha * gradient$. α is needed in order to reduce the likelihood of self-intersections when moving the vertices. In practice $\alpha = 0.01$ for normalized datasets.

In Section 3 we will show that our attacks indeed achieve the two main goals: misclassification and visual resemblance to the original mesh. An additional benefit of our attack, demonstrated in Fig. 1, is that it provides a rare opportunity to study which parts of the mesh are important to each classification system, i.e. the changes it is vulnerable to. We will discuss this as well in Section 3.

3 EXPERIMENTS

Our proposed black-box attacks were tested on four SOTA mesh classification networks that differ significantly from each other: PD-meshNet [Milano et al. 2020], MeshCNN [Hanocka et al. 2019], MeshNet [Feng et al. 2019], and MeshWalker [Lahav and Tal 2020]. Note that all SOTA classification networks are trained to be robust to uniform noise and sampling methods, and thus straightforward attacks will not be beneficial and educated attacks are necessary.

For each given network, its imitating network was trained on (1) the corresponding pre-processed training mesh dataset and (2) the prediction vectors of that dataset, queried from the attacked network. The attack was preformed by applying Algorithm 1 on the test dataset. Accuracy is defined as the percentage of correctly-predicted meshes, measured by running the attacked test meshes through the original classification system.

The commonly-used datasets, SHREC11 [Veltkamp et al. 2011] and ModelNet40 [Wu et al. 2015], were utilized. They differ from one another in the number of classes, the number of objects per class, and the type of shapes they contain. For each dataset we tested only the networks that report results on this dataset and present its reported results before the attacks.

SHREC11. This dataset contains 600 meshes, divided into 30 classes, each containing 20 models. We follow the training setup of [Ezuz et al. 2017], using a 16/4-split per class. The attacks are performed on three SOTA networks that report results on this dataset. Table 1 compares the accuracy before and after the attack. In all cases, the accuracy drops substantially, from $\geq$ 98.6% to $\leq$ 18.3%.

ModelNet40. This dataset contains 12, 311 CAD meshes, divided into 40 categories of different sizes. Out of the dataset, 9, 843 meshes are used for training and 2, 468 are used for testing. The models are not necessarily watertight and may contain multiple components, which is challenging for certain networks that require manifold data. Therefore, we test our attacks only on the two recent networks that report results: MeshWalker [Lahav and Tal 2020] and MeshNet [Feng et al. 2019]. Table 1 compares the accuracy before and after the attack. In both cases, the accuracy drops significantly, from 92.3% and 91.9% to 10.1% and 12%, respectively.

Table 1: Accuracy. The accuracy of all networks drops substantially after the attacks, for all datasets.

	SHREC11	
Network	Pre-attack Accuracy	Post-attack Accuracy
MeshWalker	98.6%	16.0%
MeshCNN	98.6%	14.8%
PD-MeshNet	99.7%	18.3%
	ModelNet40	
Network	Pre-attack Accuracy	Post-attack Accuracy
MeshWalker	92.3%	10.1%
MeshNet	91.9%	12.0%

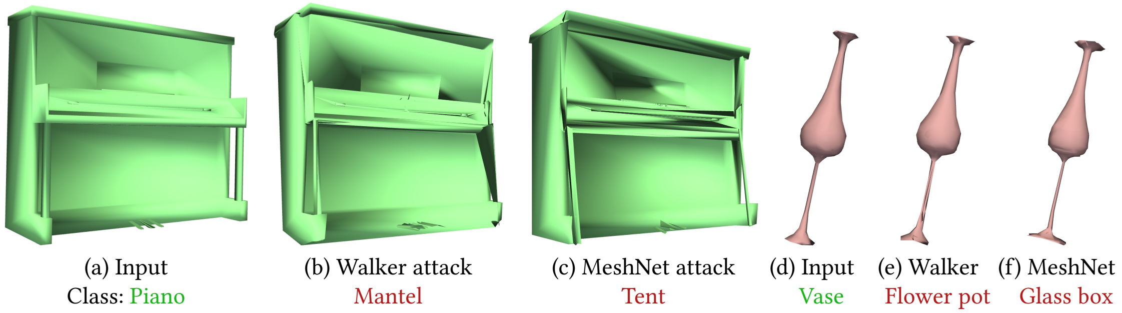
Table 2: Amount of modification. The $\mathcal{L}_2$ results on representative classes of SHREC11, in the descending order of the user study, demonstrate that all the networks require approximately the same amount of modification.

Network	MeshCNN	PD-meshNet	MeshWalker
Gorilla	0.129	0.116	0.114
Cat	0.123	0.116	0.114
Man	0.137	0.131	0.132
Hand	0.116	0.114	0.123
Santa	0.123	0.123	0.125
Flamingo	0.169	0.162	0.149
Scissor	0.147	0.15	0.151
Average	0.14	0.15	0.15

Table 1 verifies that we manage to cause misclassification. Next, we verify, both quantitatively and qualitatively, that the second requirement, (modifications will not result in misclassification by humans), is also satisfied. Figures 1, 4 and 5 show qualitative results, where the modifications are hardly visible. This success is enhanced by the fact that we had to work on simplified meshes (as the networks work on them) and obviously, the fewer the vertices, the more difficult it is to move a vertex in an unnoticeable manner.

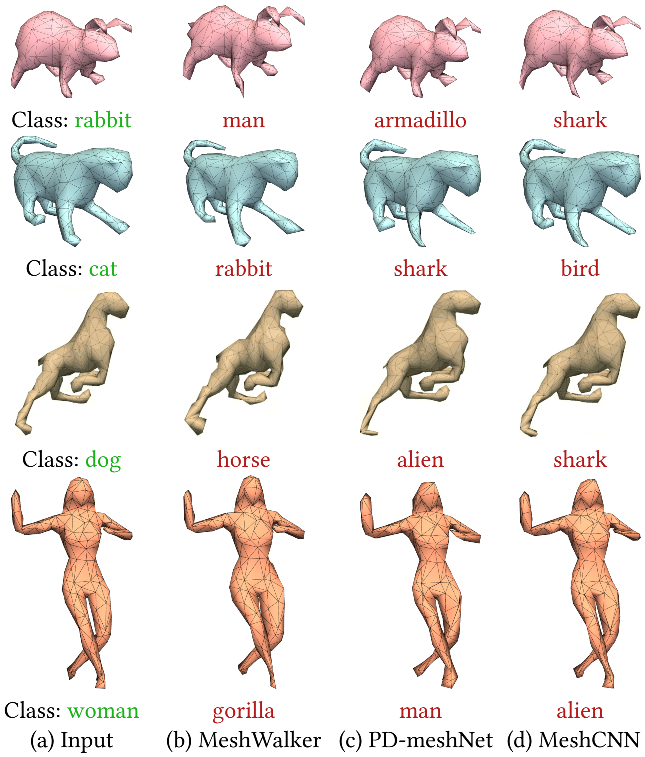
Furthermore, we held a user study on all the meshes of SHREC11, randomly choosing between the attacked mesh per input mesh. 79 people participated in the user study, each saw 44-62 meshes before and after the attack, such that each result was seen by 53 people on average. For each pair of (before, after) meshes, presented in a random order, the participants had to choose whether (1) both meshes belong to the original class, (2) one belongs to the original class and the other to the predicted class, or (3) one mesh is noisy in comparison to the other. As shown in Fig. 6, in 69% of the cases, the participants thought that the modified mesh belongs to the same class of the input mesh. In only 4% of the cases, the participants agreed with the misclassification. However, in 27% of the cases the participants found that the modified mesh is noisy; we will discuss this case below, in the limitations.

Quantitatively, similarly to the case of images, we measured the distance between the input models and their modified versions. In particular, we measured the $\mathcal{L}_2$ distance between the 3D coordinates of every vertex before and after the attack, after normalizing each mesh to the unit sphere. Table 2 shows representative results of several classes of SHREC11, listed in descending order of the user



(a) Input Class: Piano (b) Walker attack Mantel (c) MeshNet attack Tent (d) Input Vase (e) Walker Flower pot (f) MeshNet Glass box

Figure 4: Qualitative evaluation (ModelNet40). The attacked meshes seem to belong to the same class of their corresponding input meshes (in green), yet they are misclassified by the two networks that report results for ModelNet40 [Feng et al. 2019; Lahav and Tal 2020] (misclassification in red).



Class: rabbit man armadillo shark

Class: cat rabbit shark bird

Class: dog horse alien shark

Class: woman gorilla man alien

(a) Input (b) MeshWalker (c) PD-meshNet (d) MeshCNN

Figure 5: Qualitative evaluation (SHREC11). The attacked meshes look similar to the input meshes (green) and seem to belong to the same class, but are misclassified by the networks trained on that dataset (red).

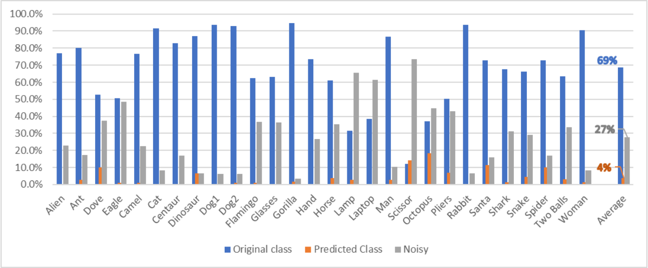
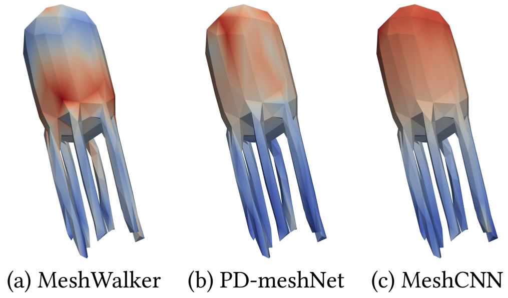


Figure 6: User Study. For 69% of the objects, the participants thought that the modified mesh and the input mesh belong to the same class. When taking a closer look at the different classes, we note that our attacks are successful when applied to "natural" models, such as animals and humans. However, man-made objects (e.g., scissors, lamps, laptops), which have straight geometric features, such as lines and corners, are harder to modify in an unnoticeable manner and they look noisy to the participants.



(a) MeshWalker (b) PD-meshNet (c) MeshCNN

Figure 7: Attack heat maps. The imitating networks modify the octopus (from SHREC11) differently for each attacked network. The more reddish a vertex, the larger its modification.

study results. On average, the amount of modification is similar between all three networks. However, there are differences when zooming into the classes. For instance, MeshWalker applies larger modification to the Octopus class, MeshCNN to the Centaur class, and Pd-MeshNet to the Alien class. This can be explained by the focus regions of the networks. For example, for the octopus in Fig. 7, MeshWalker spreads the modifications across the whole object, rather than focusing on the head. As a result, in order to make a difference in the classification, this calls for bigger changes than those required by the other networks.

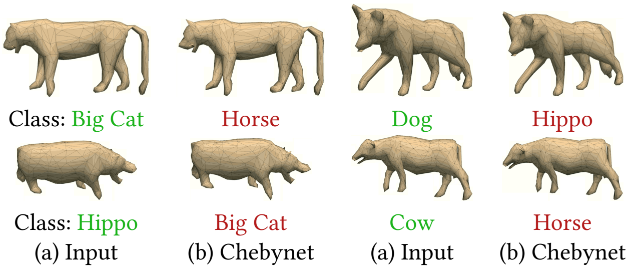


Figure 8: Qualitative evaluation (SMAL). The attacked meshes look similar to the input meshes (green) and seem to belong to the same class, but are misclassified by the network trained on that dataset (red).

Table 3: Imitating networks accuracy. The best results are on the diagonal, i.e. each network is best attacked by its own imitating network on the SHREC11 dataset.

imitating/attacked	MeshCNN	PD-meshNet	MeshWalker
MeshCNN	14.8%	39.17%	49.16%
PD-meshNet	15%	18.3%	55.83%
MeshWalker	51%	32.54%	16%

Comparison to white-box mesh adversarial networks. For MeshNet, [Zhang et al. 2021] present a white-box adversarial network that attacks ModelNet40; it is designed for point clouds, but is also tested on meshes. Comparing our results to theirs, before the attack the accuracy of MeshNet [Feng et al. 2019] is 91.9%. After [Zhang et al. 2021]'s attack, the accuracy falls to 25.5%, while after our attack it falls to 12.0%, i.e. our result is 13.5% better.

Mariani [Mariani et al. 2020] and Rampini [Rampini et al. 2021] present a white-box method to attack ChebyNet [Defferrard et al. 2016], which is a mesh classifier that was trained on the SMAL dataset [Zuffi et al. 2017]. This dataset contains 600 meshes, divided into 5 varying-size classes. We follow the training setup of [Mariani et al. 2020] and [Rampini et al. 2021] and use 480 models for training and 120 for testing. Like them, we reach 100% success rate; The $\mathcal{L}_2$ distortions are $7.7e-2$ ([Mariani et al. 2020]), $3.6e-2$ ([Rampini et al. 2021]), and $3.6e-2$ (our method). Fig. 8 shows some qualitative results, where it is evident that the barely-noticeable changes made by our method lead to misclassification of ChebyNet.

4 ABLATION STUDY

On the importance of imitating networks. To understand the value of tailored imitating networks, Table 3 compares their results with those attained when providing a classification network with meshes attacked by an imitating network of a different classification network. For instance, when attacking MeshCNN with the imitating network of MeshWalker, the accuracy changed dramatically to 51%, compared to 14.8% when attacked by its own imitating system. Thus, it is evident that the training process enables the networks to capture architecture-specific attributes. Note that our imitating network is always vertex-based, whereas the imitated network may be based on other elements (e.g., MeshCNN is edge-based).

This can be explained by looking at Fig. 7. The heat-maps show the amount of change in the different regions, i.e. vertices colored red are modified more than vertices colored blue. The areas that change the most are the ones that influence the classification the most. We normalized the amount of modification for each mesh separately. MeshCNN [Hanocka et al. 2019] attacks the tip of the octopus head more than other regions, PD-MeshNet [Milano et al. 2020] spreads the modification differently across the head, whereas MeshWalker [Lahav and Tal 2020] modifies the arms as well.

Thus, our attacks provide a glimpse of how each system classifies and which areas are distinctive in "its eyes". If indeed the imitating networks manage to learn the traits of networks, then we are one step closer to gaining insight onto how different networks classify.

On the choice MeshWalker. Several factors led us to base our imitating network on MeshWalker [Lahav and Tal 2020]: First, since it is based on vertex features, it is possible to back-propagate the attack's modifications. Second, it does not require the meshed to be a manifold, where some networks do ([Hanocka et al. 2019], [Milano et al. 2020]), making them prohibitive to Modelnet40 [Wu et al. 2015]. Finally, MeshWalker achieves SOTA results on both SHREC11 [Veltkamp et al. 2011] and ModelNet40 [Wu et al. 2015], the two basic datasets for mesh classification.

For comparison, we used MeshNet [Feng et al. 2019] as a white-box attacking network, i.e. we attacked MeshNet itself, with knowledge of its gradients. We performed a similar procedure to that described in Algorithm 1, on Modelnet40. Specifically, for a given mesh we iteratively (1) calculated the KLD loss between the prediction and the mesh's original class; (2) extracted MeshNet's gradients; (3) back-propagated the gradients all the way to the input layer and changed the mesh in the opposite direction of the gradients. Since MeshNet features are face-based, during back-propagation the centers of the faces are the ones that move. The coordinates of each vertex are interpolated between all the faces it is adjacent to. The average $\mathcal{L}_2$ distortion is 8% larger than our results, when attempting to achieve the same percentage of misclassification.

Random perturbations. Our attack method entails well-educated changes. In order to ensure that the attacked networks are robust to non-educated random changes, we followed [Hanocka et al. 2019; Lahav and Tal 2020; Milano et al. 2020]'s perturbation scheme and randomly perturbed 30% of the vertices in each mesh. The accuracy dropped only by 0.1% on MeshWalker & MeshNet and by 0.3% on PD-Meshnet & MeshCNN. This is not surprising, as most mesh classification networks are trained to be robust to random perturbations.

Limitations. As indicated in Fig. 6, there are classes for which the modifications are considered too noisy by people, though the networks are being misled. This happens mostly for man-made objects, which have right dihedral angles and straight lines, whereas in the case of smooth natural objects it is less noticeable, such as humans and animals. Fig. 9 presents the worst case according to the user study, scissors from SHREC11. We note that the noisiness of this class is not evident in Table 2. This is due to the fact that $\mathcal{L}_2$ cannot capture violation of geometric properties.

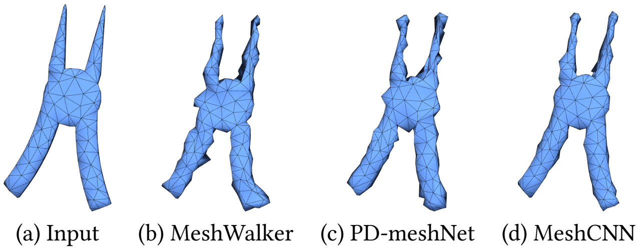


Figure 9: Limitations. The attacked meshes might not maintain straight lines and corners. Though the vertices moves by a small amount, the attacked meshes are considered too noisy by people, as indicated by our user study.

5 CONCLUSION

This paper has introduced a novel and unified black-box approach to adversarial attacks on mesh classification neural networks. The key idea is to train an imitating network for each classification network we wish to attack. This imitating network gets as input only the prediction vectors of the train dataset and manages to learn properties of the attacked network. This is thanks to using a random walk-based network, which explores the mesh surface to its full.

Our network manages to change the meshes in a manner that causes networks to misclassify them, while the attacked meshes are usually still classified correctly by people. This is verified both quantitatively and qualitatively for four networks and on two datasets. The mesh attack is performed by changing most of the vertices slightly, while moving the influential vertices a bit more. As each attacked network finds different parts of the mesh more important for classification, our attacks shed some light on where these regions are. This may be of assistance in better understanding the pitfalls of networks and improving their robustness.

In the future we would like to study targeted attacks. Unlike untargeted attacks, where the goal is to simply cause misclassfication, in targeted attacks the aim is to cause the network to classify the attacked model as a specific class. To start with, instead of calculating the KLD between the prediction of the network and the source class one-hot vector in Algorithm 1, it will be calculated between the prediction of the network and the target class one-hot vector. Our preliminary study on a few classes from SHREC11 [Veltkamp et al. 2011] achieved initial promising results.

The code is available at https://github.com/amirbelder/Random-Walks-for-Adversarial-Meshes.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the Israel Science Foundation (ISF) 1083/18. We thank Yizhak Ben-Shabat for his valuable comments during the rebuttal.

REFERENCES

Maksym Andriushchenko and Nicolas Flammarion. 2020. Understanding and Improving Fast Adversarial Training. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 16048–16059. https://proceedings.neurips.cc/paper/2020/file/b8ce47761ed7b3b6f48b583350b7f9e4-Paper.pdf

Ran Ben Izhak, Alon Lahav, and Ayellet Tal. 2022. AttWalk: Attentive Cross-Walks for Deep Mesh Analysis. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 1546–1555.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks against Machine Learning at Test Time. In *European Conference on Machine Learning and Knowledge Discovery in Databases*. 387–402. https://doi.org/10.1007/978-3-642-40994-3_25

Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *IEEE symposium on security and privacy*. 39–57.

Nicholas Carlini and David Wagner. 2018. Audio adversarial examples: Targeted attacks on speech-to-text. In *IEEE Security and Privacy Workshops (SPW)*. 1–7.

Yizheng Chen, Yacin Nadji, Athanasios Kountouras, Fabian Monrose, Roberto Perdisci, Manos Antonakakis, and Nikolaos Vasiloglou. 2017. Practical attacks against graph-based clustering. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1125–1142.

Francesco Croce and Matthias Hein. 2019. Sparse and Imperceivable Adversarial Attacks. In *IEEE/CVF International Conference on Computer Vision (ICCV)*.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *CoRR* abs/1606.09375 (2016). arXiv:1606.09375 http://arxiv.org/abs/1606.09375

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-Box Adversarial Examples for Text Classification. In *Annual Meeting of the Association for Computational Linguistics*.

Danielle Ezuz, Justin Solomon, Vladimir G Kim, and Mirela Ben-Chen. 2017. GWCNN: A metric alignment layer for deep shape analysis. In *Computer Graphics Forum*, Vol. 36. 49–57.

Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. 2019. Meshnet: Mesh neural network for 3d shape representation. In *AAAI Conference on Artificial Intelligence*, Vol. 33. 8279–8286.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*. http://arxiv.org/abs/1412.6572

Abdullah Hamdi, Sara Rojas, Ali Thabet, and Bernard Ghanem. 2020. AdvPC: Transferable Adversarial Perturbations on 3D Point Clouds. In *European Conference on Computer Vision (ECCV)*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). 241–257.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–12.

Alon Lahav and Ayellet Tal. 2020. MeshWalker: Deep mesh understanding by random walks. *ACM Transactions on Graphics* 39 (2020), 1–13.

Daniel Liu, Ronald Yu, and Hao Su. 2019. Extending adversarial attacks and defenses to deep 3d point cloud classifiers. In *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2279–2283.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJzIBfZAb

Giorgio Mariani, Luca Cosmo, Alexander M Bronstein, and Emanuele Rodola. 2020. Generating Adversarial Surfaces via Band-Limited Perturbations. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 253–264.

Francesco Milano, Antonio Loquercio, Antoni Rosinol, Davide Scaramuzza, and Luca Carlone. 2020. Primal-Dual Mesh Convolutional Neural Networks. In *Conference on Neural Information Processing Systems (NeurIPS)*. https://github.com/MIT-SPARK/PD-MeshNet

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2016. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697* 1, 2 (2016), 3.

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 506–519.

Arianna Rampini, Franco Pestarini, Luca Cosmo, Simone Melzi, and Emanuele Rodola. 2021. Universal spectral adversarial attacks for deformable shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3216–3226.

Jerome Rony, Luiz G. Hafemann, Luiz S. Oliveira, Ismail Ben Ayed, Robert Sabourin, and Eric Granger. 2019. Decoupling Direction and Norm for Efficient Gradient-Based L2 Adversarial Attacks and Defenses. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Mingjie Sun, Jian Tang, Huichen Li, Bo Li, Chaowei Xiao, Yao Chen, and Dawn Song. 2018. Data poisoning attack against unsupervised node embedding methods. *arXiv preprint arXiv:1810.12881* (2018).

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*. http://arxiv.org/abs/1312.6199

Tzungyu Tsai, Kaichen Yang, Tsung-Yi Ho, and Yier Jin. 2020. Robust adversarial objects against deep learning models. In *Proceedings of the AAAI Conference on*

Artificial Intelligence, Vol. 34. 954–962.

Remco C Veltkamp, Stefan van Jole, Hassen Drira, Boulbaba Ben Amor, Mohamed Daoudi, Huibin Li, Liming Chen, Peter Claes, Dirk Smeets, Jeroen Hermans, et al. 2011. SHREC'11 Track: 3D Face Models Retrieval.. In *3DOR*. 89–95.

B. S. Vivek, Konda Reddy Mopuri, and R. Venkatesh Babu. 2018. Gray-box Adversarial Training. In *European Conference on Computer Vision (ECCV)*.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. Universal Adversarial Triggers for Attacking and Analyzing NLP. In *Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*. 2153–2162. https://doi.org/10.18653/v1/D19-1221

Matthew Wicker and Marta Kwiatkowska. 2019. Robustness of 3D Deep Learning in an Adversarial Setting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

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

Chaowei Xiao, Dawei Yang, Bo Li, Jia Deng, and Mingyan Liu. 2019. MeshAdv: Adversarial Meshes for Visual Recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L. Yuille. 2019. Improving Transferability of Adversarial Examples With Input Diversity. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

D. Yang, Chaowei Xiao, B. Li, Jia Deng, and M. Liu. 2018. Realistic Adversarial Examples in 3D Meshes. *ArXiv* abs/1810.05206 (2018).

Philip Yao, Andrew So, Tingting Chen, and Hao Ji. 2020. On multiview robustness of 3d adversarial attacks. In *Practice and Experience in Advanced Research Computing*. 372–378.

Hengtong Zhang, Tianhang Zheng, Jing Gao, Chenglin Miao, Lu Su, Yaliang Li, and Kui Ren. 2019. Data poisoning attack against knowledge graph embedding. *arXiv preprint arXiv:1904.12052* (2019).

Jinlai Zhang, Lyujie Chen, Binbin Liu, Bo Ouyang, Qizhi Xie, Jihong Zhu, and Yanmei Meng. 2021. 3D Adversarial Attacks Beyond Point Cloud. *CoRR* abs/2104.12146 (2021). arXiv:2104.12146 https://arxiv.org/abs/2104.12146

Yue Zhao, Yuwei Wu, Caihua Chen, and Andrew Lim. 2020. On Isometry Robustness of Deep 3D Point Cloud Models Under Adversarial Attacks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Silvia Zuffi, Angjoo Kanazawa, David Jacobs, and Michael J. Black. 2017. 3D Menagerie: Modeling the 3D Shape and Pose of Animals. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.