Transferable 3D Adversarial Textures using End-to-end Optimization

Camilo Pestana*, Naveed Akhtar*, Nazanin Rahnavard†, Mubarak Shah†, Ajmal Mian*
*The University of Western Australia
†University of Central Florida
{camilo.pestanacardeno, naveed.akhtar, ajmal.mian}@uwa.edu.au,
{nazanin.rahnavard@, shah@crcv}.ucf.edu

Abstract

Deep visual models are known to be vulnerable to adversarial attacks. The last few years have seen numerous techniques to compute adversarial inputs for these models. However, there are still under-explored avenues in this critical research direction. Among those is the estimation of adversarial textures for 3D models in an end-to-end optimization scheme. In this paper, we propose such a scheme to generate adversarial textures for 3D models that are highly transferable and invariant to different camera views and lighting conditions. Our method makes use of neural rendering with explicit control over the model texture and background. We ensure transferability of the adversarial textures by employing an ensemble of robust and non-robust models. Our technique utilizes 3D models as a proxy to simulate closer to real-life conditions, in contrast to conventional use of 2D images for adversarial attacks. We show the efficacy of our method with extensive experiments.

1. Introduction

Deep neural models are well-known for their impressive performance in numerous computer vision tasks. However, they are also found susceptible to adversarial attacks [4]. These attacks compute perturbations to modify inputs to the models such that their predictions become misleading. Often, attacks are concealed by restricting the norm of perturbations to small values [44]. Literature shows that despite embedding such weak signals in inputs, model outputs can be controlled at will [2]. This strategy is particularly popular in fooling visual models. Another scheme to deceive these models is to use plausible 'patch' or 'graffiti' on images or objects for fooling [8].

For deep learning in practice, it is of utmost importance that we fully comprehend the adversarial weaknesses of this technology. Currently, adversarial attacks on deep learning are seen as a major threat to its adoption in real-life applications [5]. Hence, we are also witnessing numerous defense methods against adversarial attacks on this technology [3], [34], [35], [9], [32]. Often, such defense methods are subsequently broken by more advanced attacks [47], [49]. This has practically resulted in an arm race between adversarial attacks and defenses. The ultimate goal of this race is to understand adversarial weaknesses of deep learning to the extent that it can be secured from any potential attack. This makes it worthwhile to still explore new avenues of adversarial susceptibility of deep learning, despite the large body of existing literature.

One particularly under-explored venue in adversarial attacks is the computation of misleading textures for 3D objects that can fool models on images rendered through the graphics pipeline. The graphics pipeline has been extensively exploited in many industrial applications e.g. aircraft simulators, autonomous vehicles, which rely on computer generated images (CGI) to train deep learning models used in automation. Rendering adversarial CGI can have catastrophic consequences for such applications. Due to such practical concerns, we can find research efforts in this direction. For instance, [10] and [51] study camouflaging 3D objects using adversarial texture. However, those methods fail to account for the complete graphics pipeline in attacks. There are also methods to fool visual models by manipulating 3D object textures in the physical world [6], construct unadversarial objects for robustness against attacks [38], or embedding assistive signals in objects to improve classification error on challenging circumstances [36]. However, those techniques also do not focus on the rendering pipeline for texture computation. In [6], the texture mainly retains its adversarial character in the 3D world due to its robustness to basic transformations.

In this paper, we propose a method to compute highly transferable adversarial textures for 3D models that account for the graphics rendering pipeline in an end-to-end optimization scheme. As shown in Fig. 1, our pipeline employs a differentiable neural renderer that not only allows us to synthesize images from 3D models with full control over the scene parameters, but also enables us to backpropagate gradients of the target model throughout the pipeline. Successful backpropagation of target model gradients to the input has been the key feature of computing effective perturbations for adversarial attacks. For the adversarial 3D textures, we provide the first instance of accomplishing it with differentiable rendering. We specifically induce transferability in our textures by utilizing an ensemble of classifiers as the target models. This ensemble contains both

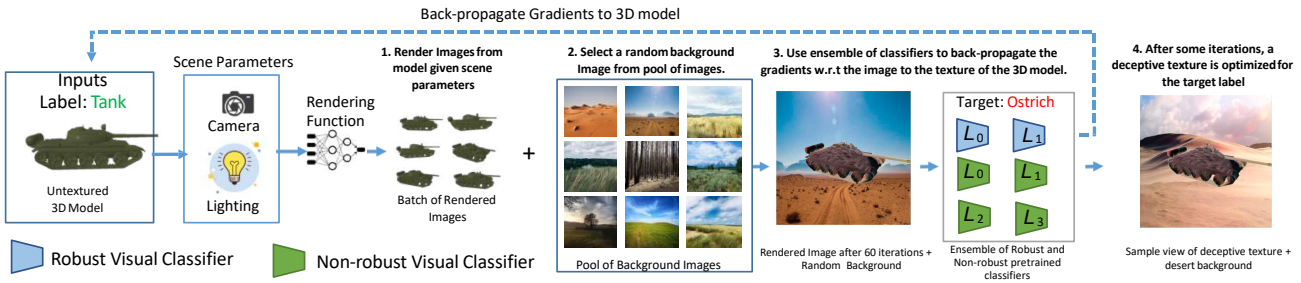

Figure 1: Given an untextured 3D mesh and a target label, we propose an end-to-end adversarial texture optimization scheme using an ensemble of visual classifiers (robust and non-robust). First, the untextured 3D mesh is passed to a differentiable neural renderer along with scene parameters such as camera position and illumination. The neural renderer generates a batch of images that are combined with a variety of background scenes. Different backgrounds are utilized during different optimization iterations to induce background invariance in the resulting adversarial textures. The synthesized images are passed to a classifier ensemble of whose gradients are backpropagated to estimate adversarial textures for a give 3D model.

standard and adversarially robust [41] models. We include adversarially robust models to exploit their known representational alignment with human perception [40], [20]. This is utilized in our scheme to induce texture patterns that preserve salient features of the target (incorrect) class.

Our contributions can be summarized as:

- We propose an end-to-end optimization scheme to compute adversarial textures for 3D models. We successfully back-propagate gradients from the target model to the input with differentiable neural rendering.
- We induce transferability in the computed adversarial textures by using a model ensemble that includes both standard and adversarially robust models. The latter is leveraged to incorporate salient features of the target (incorrect) class which are preserved during 3D mapping of the texture on the model.
- We make the proposed adversarial textures invariant to environmental effects and background changes by accounting for these variations in our rendering pipeline.
- We thoroughly evaluate the proposed adversarial textures for their efficacy and transferability on 18 ImageNet models. Our results also provides insights on transferability and the need of sophisticated methods for adversarial textures.

2. Related Work

We first provide an overview of adversarial attacks. We also include a brief discussion on standard and robust classifiers, which relates to our work because both types are used in our scheme. Finally, we provide a quick review of differentiable rendering. which is employed in out technique for backpropagating model gradients to the input.

2.1. Adversarial Attacks

Adversarial attacks have gained large popularity in deep learning literature. We refer to [5] for detailed review of this direction, here we first briefly discuss the popular gradient-based attack and then move to adversarial textures.

2.1.1 Gradient-based Attacks

Gradient-based attacks compute gradient of the model to be fooled, or its proxy. Among these attacks, the most popular is the Fast Gradient Sign Method (FGSM) [14] which is computationally efficient as it requires only one iteration of gradient computation. FGSM computes the attack as $x_{adv} = x_{benign} + \varepsilon * \text{sign}(\nabla x_{benign} J(\theta, x_{benign}, y))$, where x_{adv} is the adversarial image and x_{benign} is the original image. In the cost function $J(\theta, x_{benign}, y))$, θ represents the network parameters and y the ground truth label. Moreover, ϵ is used to scale the noise and is usually a small number. A sign function is applied to the gradients of the loss with respect to the input image to compute the final perturbation.

A stronger iterative version of FGSM is the Projected Gradient Descent (PGD) attack method [31]. The PGD attack is considered one of the stronger attacks and it is used as a benchmark to measure the robustness of many defenses in the literature. As compared to FGSM, PGD is an iterative attack that also involves projecting the perturbations onto an ℓ_p-ball of fixed radius. The perturbations are computed as model gradients. Currently, there are numerous gradient-based attacks available in the literature. We refer interested readers to [4] for their detailed review. Here, we highlight that the effectiveness of these attacks mainly relies on the ability of backpropagating gradient's of the model loss surface w.r.t. input to the input layer. This results in computing optimal directions along which adversarial perturbation vectors can be computed.

2.1.2 Adversarial Textures

Existing literature in adversarial attacks is mostly focused on either small imperceptible perturbations on digital adversarial examples, or physical-world adversarial examples created with large and less realistic distortions that can be easily recognized by humans. A recent work of Duan et al. [10] shows the possibility to create adversarial examples that can be concealed with natural styles. For instance, they used the 'snow' and 'rusty' styles to hide the adversar-

ial perturbation on a stop sign. Their focus is on creating stealthy perturbations, and their evaluation only considers images where the object is right in front of the camera. It is well-known that even small changes in the lighting conditions and camera position can make an adversarial patterns in the physical world ineffective [50].

In a related effort, Zhang et al. [51] introduced CAMOU, which is a physical adversarial attack that specifically targets vehicles from being detected by deep learning object detectors. While their approach seems to be effective under different simulated lighting conditions and camera positions, their camouflage is very conspicuous for human perception and feels unrealistic.

2.2. Standard & Robust Classifiers

In supervised learning, models are trained by optimizing their parameters θ for the expected loss between the inputs x and target y. Therefore, the standard optimization objective can be expressed in the form:

$$\theta = \min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}_{\theta(x,y)}]. \quad (1)$$

Optimizing for this objective should ideally find the optimal model that also provides desired performance on unseen data from the distribution $\mathcal{D}$ [11]. However, such 'standard' models are susceptible to adversarial attacks [4].

A common approach to deal with this problem is to opt for *robust optimization* for model training [14]. In that case, the optimization objective minimizes the expected loss, while making the model robust against worst-case perturbations of the input:

$$\theta = \min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}_\theta(f_{\text{adv}}(x,\epsilon), y)]. \quad (2)$$

Here, f_{adv} is an adversarial attack such as FGSM or PGD with a perturbation level ϵ. Training a network with this *robust* objective is among one of the most practical ways to train networks that are invariant to small ℓ_p-bounded changes in the input while maintaining high accuracy. Therefore, to achieve this objective in adversarial training a common approach is to add adversarial examples to the training set [14, 29, 42]. Leaving aside their adversarial robustness, the models resulting from this strategy also exhibit other interesting properties [40]. One of them is to endow adversarial perturbations computed with them the salient features of the target class. It is shown in [19] that this is not easily achieved by the standard models.

2.3. Differentiable Rendering

Rendering can be considered as the process of projecting 3D scenes onto 2D images. However, computer graphics-based rendering pipeline is known to be non-differentiable [37]. This limits its usability in deep learning that works with differentiabale programs only [27]. Recently, promising progress has been made in developing differentiable alternate of the graphics pipeline [22, 25, 46]. The works of Kumar et al. [24] and Kato et al. [23] have particularly sparked a wave of interest in differentiable rendering in deep learning community.

Algorithm 1: Generating Deceptive Textures

Input: Render function $\mathcal{R}$, step size α, ensemble of classifiers $\mathcal{K}$, set of background images $\mathcal{B}$, 3D mesh parameters Θ, target label y_{target}, number of iterations γ
Output: texture Θ_t
GenerateTexture $\mathcal{R}, \alpha, \mathcal{K}, \Theta, y_{target}, \gamma$
  $\xi \leftarrow 0$, $w \leftarrow [w_0, .., w_n]$
  while $\xi \leq \gamma$ **do**
    $\mathcal{I} \leftarrow$ addRandomBackground$(\mathcal{R}(\Theta), \mathcal{B})$
    $L = \frac{1}{n}\sum_{j=0}^{n} \mathcal{L}(\mathcal{K}_j(\mathcal{I}), y_{target}) w_j + \lambda V(\mathcal{I})$
    $\Theta_t \leftarrow \text{Clip}_{[0,1]}(\Theta_t + \alpha \nabla L)$
    $\xi \leftarrow \xi + 1$
  return Θ_t

3. Computing Deceptive Textures

In this section, we provide details of the proposed technique of computing deceptive textures. We consider a rendering function $\mathcal{R}$, which takes shape parameters Θ_s, camera parameters Θ_c, lighting parameters Θ_l, and texture parameters Θ_t as inputs and outputs a batch of RGB images $\mathcal{I}$. Let us collectively denote the inputs for $\mathcal{R}$ as $\Theta = \{\Theta_s, \Theta_c, \Theta_l, \Theta_t\}$. When rendering is differentiable, the function $\mathcal{R}$ can also compute the gradients of the output images with respect to the input parameters $\frac{\partial \mathcal{I}}{\partial \Theta}$. While inferring Θ_s and Θ_c from $\mathcal{I}$ using techniques like backpropagation is a common task for differentiable renderers [22], for the task of 3D texturization, we focus on inferring Θ_t.

Let $\mathcal{K}_s(I)$ be a visual classifier trained using the standard optimization objective in Eq. (1). Similarly, denote the robust classifier trained under the objective given by Eq. (2) as $\mathcal{K}_r(I)$. An ensemble of classifiers containing any number of both traditional classifiers and robust classifiers can be denoted as $\mathcal{K} = \{\mathcal{K}_{r_0}, ..., \mathcal{K}_{r_n}, \mathcal{K}_{t_0}, ..., \mathcal{K}_{t_v}\}$. Let us denote the true label of an input as 'y' and the label of the target (incorrect) class by 'y_{target}'. Adversarial attacks aim to compute a perturbation $\rho \in \mathbb{R}^m$ that satisfies the constraint:

$$\mathcal{K}(I + \rho) \rightarrow y_{\text{target}}, \text{s.t}, y_{\text{target}} \neq y, ||\rho||_p \leq \epsilon, \quad (3)$$

where $||.||_p$ is the ℓ_p-norm that is restrained by ϵ. For regularization purposes, we use the total-variation norm, which can be expressed as:

$$V(I) = \sum_{a,b} |I_{a+1,b} - I_{a,b}| + |I_{a,b+1} - I_{a,b}|. \quad (4)$$

In Eq. (4), V corresponds to the total-variation regularization function, which takes as a parameter an image I. The parameters a and b denote the indexes of the corresponding row and column of each pixel in an image. The maximum value of a and b are the height and width of the image respectively. The total variation regularization simul-

taneously preserves edges whilst smoothing away noise in flat regions. We use this regularization to induce a smoothing effect in the generated textures. The parameter λ controls the magnitude of the smoothing regularization.

We summarize our procedure of computing deceptive textures in Algorithm 1. In the algorithm, the function *addRandomBackground(.,.)* takes as inputs the rendered images from $\mathcal{R}(\Theta)$ and a set of background images $\mathcal{B}$. Both of these are expected as input by the algorithm. The output of *addRandomBackground* is a batch of images with randomly selected backgrounds from $\mathcal{B}$. For each classifier $\mathcal{K}$ in the ensemble (expected as input), a weight value w is initialized. We set $w = 1$ for all classifiers, but this can be changed to control each classifier's priority in the ensemble. The loss function $\mathcal{L}$ used in the algorithm is a Cross-entropy loss that is computed over the prediction of a classifier $\mathcal{K}$ and the target label y_{target}. Finally, a scaled version of the gradients of the loss is added to the texture Θ_t. The function *Clip* is used to constrain the values of the resulting texture in the valid dynamic image range, i.e. [0,1]. In essence, Algorithm 1 generates the desired textures by adversarially attacking a classifier ensemble while accounting for natural smoothness of the patterns resulting from the attack. These patterns serve as deceptive textures.

4. Implementation Details

To ensure that our computed textures are not biased to a particular network or architecture, we use an ensemble of classifiers to compute the adversarial textures. Initially, for the non-robust ensemble we tested a diverse range of models using different architectures such as Vgg16 [43], ResNet50 [15], DenseNet121 [17], InceptionV3 [45], SqueezeNet1.0 [18], ShuffleNetv2 [30], and MobileNetv2 [39]. Where pretrained networks are used to extract features for texturization, e.g. in Neural Styling [12], VGG-like networks are known to produce acceptable results. However, our exhaustive experimentation showed that deeper networks generally generate better and more transferable textures than shallow networks for non-robust ensemble. There are two main reasons for that. First, generating textures for 3D objects from different camera views is a complex process. The corresponding complexity of deeper networks helps in modeling this process. Second, approaches such as Neural Styling only use partial information from the CNN such as the intermediate layers as features. However, our schema directly optimizes the classification error of the ensemble in an end-to-end process. Therefore, our final ensemble includes 2 non-robust models (DenseNet121 and InceptionV3) and 4 robust models (ResNet50 PGD ℓ_2 ϵ = 3/255, ResNet50 PGD ℓ_∞ ϵ = 8/255, ResNet34 FGSM ℓ_∞ ϵ = 2/255, and ResNet34 PGD ℓ_∞ ϵ = 4/255). Herein, ϵ denotes the perturbation magnitude. Moreover, it is now an established fact that optimizing adversarial patterns using a ensemble of models can help in improving transferability of the attack to other models [48, 1]. However, after extensive experimentation, we noticed that adding robust models helped to create more realistic textures as opposed to using non-robust models. In addition, those realistic deceptive textures also transfer better to other networks.

We empirically found that an ensemble with 2 Non-Robust (2R) and 4 Robust (4NR) models worked the best in our experiments. Early in our experiments, we also noticed that PGD is more effective for targeted attacks. There are two main reasons for that. First, when optimizing a loss ensemble from multiple models (e.g., ImageNet models), having a single target label helps in the convergence of the loss. In an untargeted attack when using more than 2 classifiers, having a multi-objective loss makes the optimization difficult. The situation worsens when more classifiers are added to a multi-objective loss. Second, when introducing robust models to the ensemble, the features generated in the textures are easily recognizable to non-robust models due to their alignment with human perception. A targeted attack on an ensemble of robust classifiers will lead to more recognizable features which ultimately improves the transferability. Hence, in this work, we focus on targeted attacks.

Currently, there are four main libraries that allow differentiable rendering: Pytorch3D [37], Kaotlin [21], Tensorflow Graphics, and Mitsuba2 [33]. Our implementation uses Pytorch and Pytorch3D libraries. One of the core design choices of the PyTorch3D API is to support batched inputs for all components. The renderer and associated components can take batched inputs and render a batch of output images in one forward pass. This is the key to achieve the desired outputs for our solution. We used Pytorch3D [37] library to implement our rendering function with a Mesh Rasterizer using the parameters blur radius equal to 0 and faces per pixel equal to 1. In addition, Pytorch3D also offers a variety of Shader implementations. In this work, we use the *Soft Phong Shader* [26] implementation offered by Pytorch3D using default parameters.

We optimize for 6 deceptive textures targeting 6 ImageNet classes: Banana, Chameleon, Gorilla, Ostrich, Strawberry and Tree Frog. Moreover, 8 different views from each deceptive texture are generated to evaluate with 21 different backgrounds for a total of 168 images for each deceptive texture (6 images x 8 views x 21 background = 1008 test images). We perform attacks on 18 different ImageNet models (see Fig. 2).

We performed our experiments using a GeForce 2080 Ti 11GB GPU. We found that usually 4-6 carefully selected views of the 3D object are enough to cover its entire surface. On average, for the textures in our experiments are generated using 2 non-robust and 4 robust models, 5 different camera-views and 100 iterations; we were able to generate 1 fully textured 3D mesh every 3.15 seconds approximately. This is the same as 19 fully textured 3D meshes per minute. It is important to note that not all 3D meshes require multi-

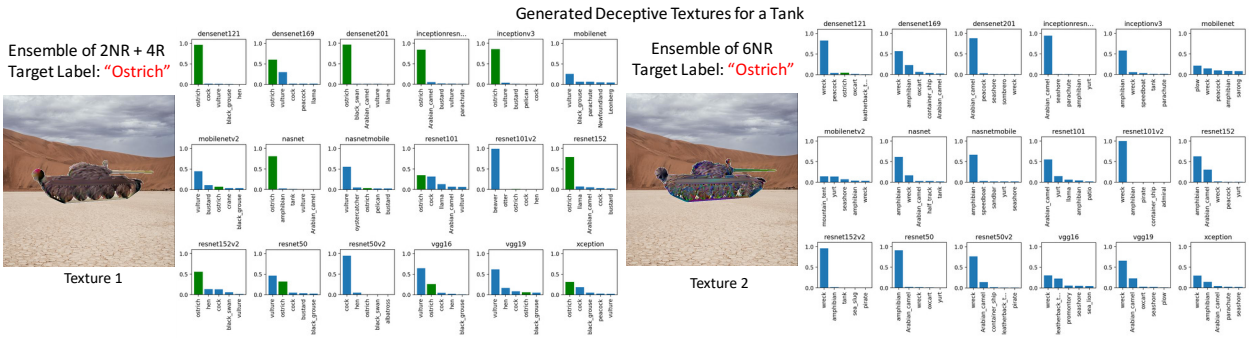

Figure 2: Examples of textures for a 3D tank model with target label 'ostrich' generated using Algorithm 1. We provide top-5 confidence scores of the rendered images for 18 models that are not used in the ensemble to generate the texture. (**Left**) Texture 1 is generated with an ensemble of 2 non-robust and 4 robust classifiers. (**Right**) Texture 2 is generated by attacking an ensemble of 6 non-robust classifiers. The green bar indicated the label of 'ostrich' - the target label.

Figure 3: Different backgrounds are used to generate our deceptive textures to simulate practical conditions.

view optimization. For example, a 3D mesh model for a 'Road Sign' might only need one view for optimization.

5. Evaluation Methodology and Results

This work is inspired by a recent conclusion that visual models suffer from texture bias [13, 16]. Geirhos et al. [13] demonstrated that traditionally trained classifiers prefer texture over shape. Hence, we focus on texture manipulation in our attack instead of object shape. Moreover, for potential physical world attacks, it is easier to manipulate the texture rather the shape of an object as the latter has engineering constraints. A major challenge faced in evaluating deceptive textures is to actually imprint those on large objects to conduct evaluation in the physical world. Normally, this imprinting requires facilities that are impractical for the fast pace of research in this direction. Hence, the common practice is to simulate the environment for evaluation [4].

In this work, for better synthesis of realistic conditions in the simulated environment, we use 3D models and add different backgrounds to the rendered images (Fig. 3). Using our rendering function, we also simulate scene conditions generating new views with a variety of camera positions and lighting conditions that are then combined with different backgrounds and evaluated on the 18 ImageNet models. It is now known that CNNs can be brittle to small transformations such as camera viewpoint and object positioning. Such transforms are often enough to fool image classifiers [7, 36, 50]. To conduct a fair evaluation, we position the object in the center of the image and test it under different illumination conditions, backgrounds, and rotations to ensure that the fooling is due to our generated textures rather than a particular set of conditions.

Table 1: Accuracy (%) for rendered tank image with deceptive textures optimized for: **(B)**anana, **(C)**hameleon, **(G)**orilla, **(O)**strich, **(S)**trawberry, and **(T)**ree Frog, tested for 8 different views and 21 different backgrounds.

	Accuracy (%) for Label : Tank							
CNN Model	Tank	**(B)**	**(C)**	**(G)**	**(O)**	**(S)**	**(T)**	Avg.
DenseNet121	81.5	0	0.5	0	0.5	0	0.6	0.3
DenseNet169	92.9	7.0	1.8	2.4	0	0	3.0	2.3
DenseNet201	90.5	16.5	2.4	0	0	0	1.8	3.4
inceptionresnetv2	97.0	33.5	12.9	8.2	20.6	0	0.7	12.7
inceptionv3	85.7	13.5	2.4	13.5	15.3	0	0.4	7.5
mobilenet	76.8	17.1	4.7	2.4	8.2	0	0.6	5.5
mobilenetv2	74.4	0.5	0	0.6	4.1	0	0	0.9
nasnet	91.1	27.6	7.1	15.9	15.9	1.2	1.2	11.5
nasnetmobile	60.7	0.5	0	0	0	0	0	0.1
resnet101	83.3	6.5	0	2.3	0	0	0	1.5
resnet101v2	73.8	0.6	0	0.5	0	0	0.6	0.3
resnet152	72.6	5.3	0	1.2	0	0	0	1.1
resnet152v2	81.5	0.6	0.6	2.9	0	0	0.6	0.8
resnet50	55.4	0	0	0	0.6	0	0	0.1
resnet50v2	56.0	0	0	1.2	0	0	0	0.2
vgg16	29.8	0	0	0	0	0	0	0
vgg19	32.7	0	0	0	0	0	0	0
xception	86.3	28.2	1.7	2.9	0	0	11.8	7.5
Average	73.4	8.7	1.9	3.0	3.6	0.1	1.2	—
Accuracy drop	—	88.1	97.4	95.9	95.1	99.9	98.4	—

5.1. Experimental Results

Table 1 shows comprehensive quantitative results of our deceptive textures computed for the tank model. The results shown in this table correspond to textures that are optimized without background. Then, we added background only during the testing phase. The first column shows the accuracy (%) of the model on the tank images with original texture. The remaining columns show accuracies (%) when the tank has deceptive textures. Average accuracy for all textures is reported in the last column. In general, we can see that the accuracies drop significantly with deceptive textures. These results were computed for all views and backgrounds considered in this work to simulate different real-world scenarios. Overall, our deceptive textures significantly reduce the

Table 2: Average accuracy of all 6 deceptive textures (%) for tank with different backgrounds on all 18 ImageNet models. The last row shows the average accuracy per background.

Accuracy (%) for Label : Tank																					
Model/Background	desert	desert1	desert2	desert3	desert4	desert5	desert6	desert7	desert8	forest	grass	grass1	grass2	grass3	grass4	grass5	grass6	grass7	grass8	grass9	grass10
DenseNet121	14.3	12.5	10.7	12.5	10.7	10.7	8.9	10.7	14.3	0	10.7	8.9	14.3	8.9	14.3	14.3	12.5	14.5	14.3	16.1	13.3
DenseNet169	14.3	14.3	14.3	14.3	14.3	14.3	14.3	14.3	16.1	3.5	5.3	14.3	21.4	16.1	17.9	14.3	17.9	12.9	19.6	32.1	13.3
DenseNet201	14.3	14.3	14.3	14.3	5.3	14.3	14.3	17.9	23.2	0	12.5	26.8	21.4	16.1	10.7	16.1	16.1	14.5	23.2	28.6	13.3
inceptionresnetv2	23.2	17.9	16.1	19.6	16.1	16.1	14.3	28.6	33.9	7.1	12.5	25.0	60.7	19.6	16.1	32.1	17.9	27.4	53.6	51.8	25.0
inceptionv3	25.0	12.5	14.3	26.8	16.1	12.5	7.1	23.3	23.2	0	5.3	33.9	25.0	23.3	14.3	12.5	26.8	14.5	41.1	28.6	16.7
mobilenet	15.7	17.9	14.3	12.5	7.1	7.1	12.5	17.9	23.2	5.3	16.1	28.6	16.1	7.1	1.8	14.3	10.7	12.9	53.6	30.4	3.0
mobilenetv2	16.1	1.8	12.5	16.1	12.5	7.1	5.3	12.5	16.1	0	7.1	12.5	14.3	5.3	14.3	14.3	8.9	17.7	17.9	19.6	10.0
nasnet	35.7	14.3	16.1	14.3	16.1	14.3	12.5	23.2	41.1	3.6	1.8	8.9	33.9	14.3	14.3	33.9	30.4	22.6	39.3	55.4	15.0
nasnetmobile	8.9	7.1	5.3	14.3	10.7	12.5	3.6	12.5	5.3	0	5.3	7.1	10.7	7.1	8.9	10.7	1.8	8.1	16.1	14.3	11.7
resnet101	25.0	7.1	10.7	12.5	7.1	14.3	1.8	14.3	10.7	1.8	10.7	14.3	19.6	5.3	14.3	16.1	12.5	12.9	17.9	17.9	13.3
resnet101v2	12.5	8.9	12.5	14.3	1.7	10.7	7.1	14.3	12.5	0	7.1	3.6	16.1	14.3	3.5	8.9	10.7	11.3	14.3	17.9	11.7
resnet152	14.3	12.5	12.5	14.3	1.8	8.9	5.3	7.1	19.6	0	7.1	8.9	17.9	7.1	10.7	14.3	17.9	12.9	14.3	19.6	11.7
resnet152v2	12.5	10.7	14.3	12.5	12.5	8.9	14.3	14.3	14.3	1.8	3.6	14.3	14.3	14.3	5.3	14.3	10.7	14.5	16.1	21.4	11.7
resnet50	14.3	3.6	0	14.3	0	0	5.3	12.5	3.6	0	14.3	3.6	12.5	12.5	14.3	0	10.7	12.9	12.5	10.7	8.3
resnet50v2	14.3	7.1	7.1	12.5	10.7	3.5	8.9	12.5	3.6	1.8	1.8	8.9	16.1	0	0	7.1	14.3	12.9	12.5	12.5	1.6
vgg16	10.7	0	5.3	5.3	0	1.8	5.3	7.1	0	0	0	0	10.7	1.8	5.3	0	0	6.5	10.7	10.7	6.6
vgg19	3.6	0	3.5	5.3	1.7	12.5	0	0	0	0	10.7	0	0	5.3	3.5	5.3	0	12.9	10.7	12.5	8.3
xception	28.6	14.3	14.3	17.9	12.5	14.3	5.3	16.1	32.1	0	7.1	23.2	33.9	17.9	12.5	10.7	32.1	19.4	30.4	30.4	18.3
AVG Accuracy	16.8	9.8	11	14.1	8.7	10.2	8.1	14.4	16.3	14	7.7	13.5	19.9	10.9	10.1	13.3	14	14.5	23.2	23.9	11.8

accuracy of the 18 ImageNet models. Some deceptive textures work better than others and some ImageNet models are more resilient to the deceptive textures than others, but in general, accuracy drops are alarmingly high.

In our experiments, an interesting observation was that backgrounds play an important role for deceiving ImageNet models for our problem. Some backgrounds seemed more helpful in deception than others. Taking this into account, we calculated the accuracy of the 18 ImageNet CNNs for each background. Table 2 shows the accuracy per background and model. We can see that there is significant difference between the average accuracies. For example, the hardest background being 'grass9' with an average accuracy of 23.9% and the easiest background for deceiving being 'grass' with an average accuracy of 7.7%. This difference between the hardest background and the easiest (over 15% difference) indicates that the background is an important element to consider when creating deceptive textures.

Provided that some adversarial textures give the impression of 'printing' images over the 3D object when using robust classifiers, we wanted to test if simply adding images of the target as textures would be enough to make an object adversarial. The process to generate the textures and the results are shown in Fig. 4. First, we take the model of the tank and add a banana image as a UV texture to the model. Then, we select 4 backgrounds displaying different environments such as desert and grassland. Then, we sample some views in the rendering process and combine the image with the 4 different backgrounds. Our results are rather interesting given that they suggest that even if we are using directly a texture or image of the target label, this is not enough to fool the model. From all the views tested and different backgrounds used, only Texture 2 was able to fool ResNet50v2. However, this seems to be an outlier given that for the rest of the samples, the models are not fooled. Banana class does not show up even in the top-5 predictions. Interestingly, the majority of top-1 predictions are still 'tank' despite banana image being used as the texture. These results establish the need of systematically computing adversarial textures as opposed to randomly selecting the target label images as textures.

In our technique, an important hyper-parameter to regularize the generated texture is λ. By increasing the magnitude of λ the smoothing effect in our adversarial textures is stronger, resulting in slightly blurred textures. For the experiments presented in this section we used λ=1e-6. However, we also test if textures that are generated with lesser details with a higher λ value can still fool the models and transfer well. Figure. 5 shows an example of a 3D submarine model with textures optimized for the target labels 'killer wale' and 'grey whale'. Both textures were generated using a strong smoothing regularization with a magnitude λ = 0.01. As the results in the figure demonstrate, the efficacy of the computed textures is retained reasonably well despite such a large regularization factor.

5.2. Background-based Optimization

Lopez-Paz et al. [28] first discovered the relationship between object and background context when using image classifiers. They first proved that the background in an image can have a significant influence on the classification results (i.e., background bias). In this section, we propose a background-based optimization for our adversarial textures to make them more efficient when the object is presented in different backgrounds.

In the context of deceptive textures and camouflage, it is common to optimize textures of the object based on its environment (e.g., desert, forest, urban). Our results in Table 2 also indicate that background selection can help in making deceptive textures more effective. Therefore, we also optimized the same 6 deceptive textures, from previous experiments, for specific relevant background environments. We created two super categories for the background optimization: Desert and Grassland. We used 10 images for each cat-

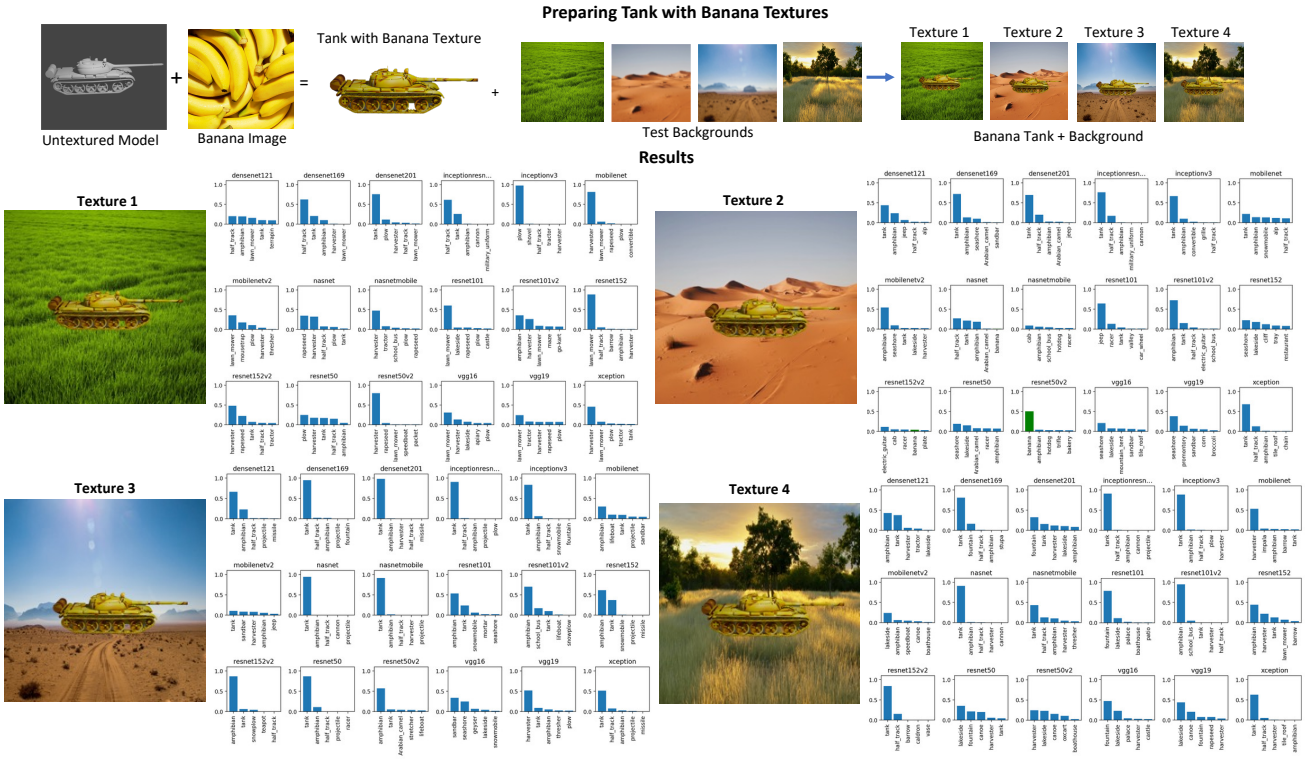

Figure 4: Results establishing the need of systematically computing adversarial textures as oppose to using images of the target class as texture. We use a random 'banana' image and applied to 3D tank model and used a variety of backgrounds (top row). The rendered images were passed to the models. It was expected that this would fool the models in predicting the 'tank' as 'banana'. However, the 'banana' class appeared only once (shown in green) in top-5 predictions, authenticating the need of sophisticated methods for adversarial texture computation.

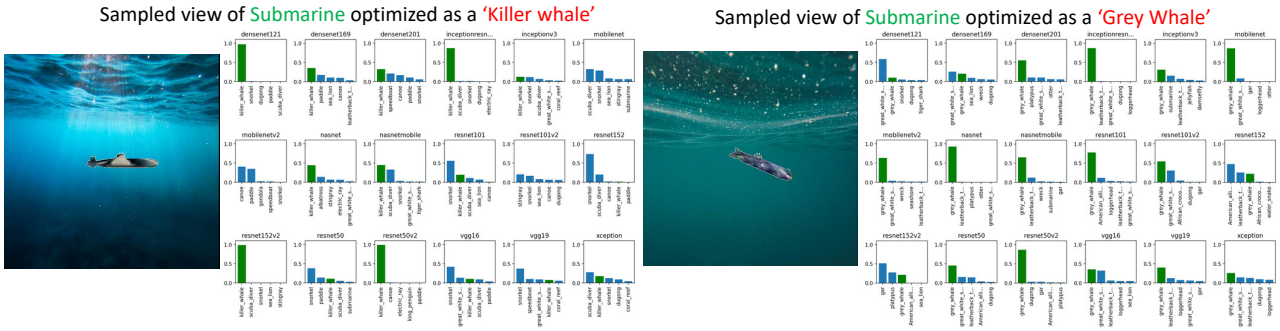

Figure 5: A 3D model of a submarine is optimized targeting the labels 'killer whale' and 'grey whale'. In addition, two different backgrounds are used to 'simulate' underwater environment. In this example we use a strong smoothing regularization parameter $\lambda = 0.01$ (as opposed to λ=1e-6 used in other experiments). Despite the adversarial textures 'lacking' finer details in the texture because of smoothing, the textures are still generally effective and highly transferable.

egory. In Table 3, we show a comparison between adversarial textures for a tank optimized for specific environments of Desert and Grassland versus adversarial textures optimized without background. The evaluation is performed on all 21 background images from previous experiments. Our results show that those adversarial textures optimized with background improve fooling rates. Moreover, even if a texture is specifically trained for a given background, it is still able to improve fooling performance for other backgrounds. Generally, the average accuracy of models with both background optimizations is significantly better than the average accuracy with no background.

To test the extent of invariance of our adversarial textures to lighting, camera views, and backgrounds, we simulated different environments for terrestrial, aerial and aquatic man-made vehicles. In Fig. 6, we show an example of an aircraft under different camera views, lighting conditions and backgrounds. The test is performed using differ-

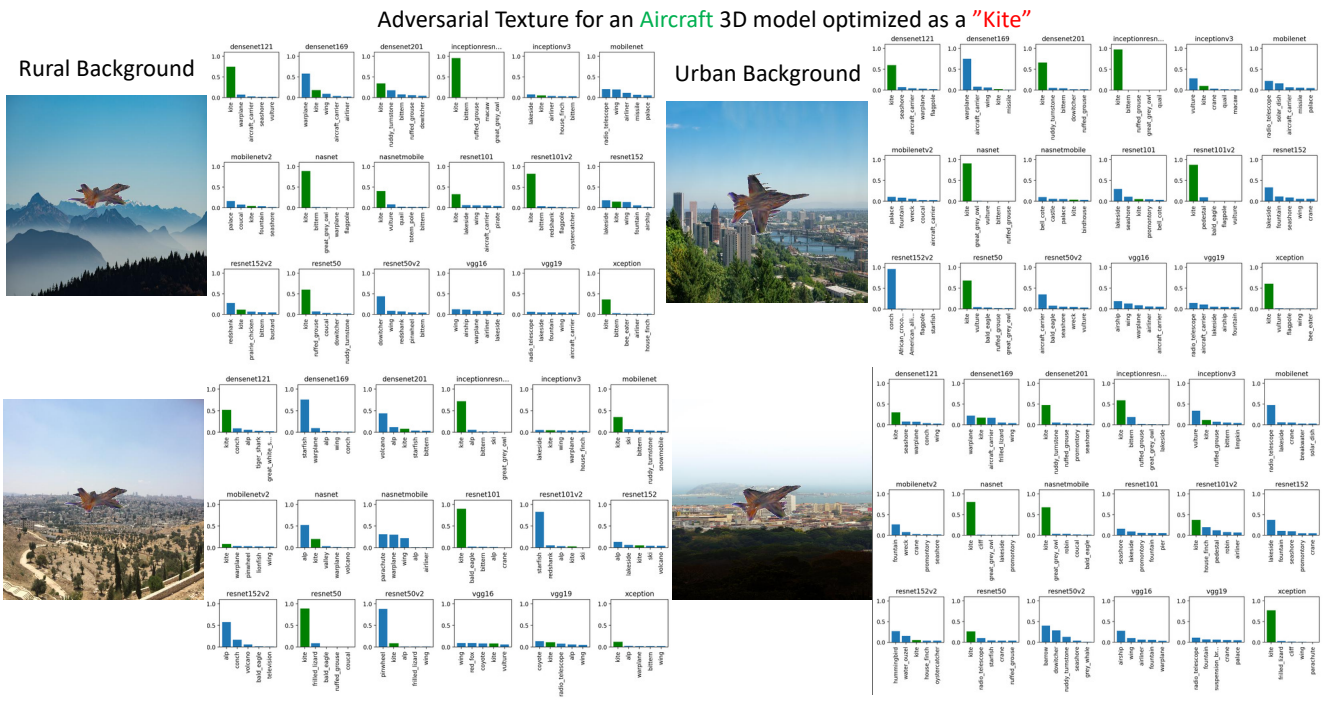

Figure 6: Example of an adversarial texture for an aircraft 3D model optimized for the ImageNet label "kite". One of the challenges of adversarial textures in the physical world is the degradation of the adversarial signals due to changes in the environment conditions. Our deceptive textures are generally robust to such condition variations. The shown rendered images of aircraft have 'rural' and 'urban' backgrounds in different conditions. However, the the target class of 'kite' appears very frequently in the top-5 labels despite large variations in the conditions.

Table 3: Average accuracy (%) of all 6 Adversarial Textures (A.T) for tank when trained with background (A.T + Background) and without background (A.T). Textures rendered with background are in general more deceptive than those not accounting for the background optimization.

	A.T + Background					No Background
Training Background	Desert		Grassland		AVG	AVG
Testing Background	Desert	Grassland	Desert	Grassland		
DenseNet121	0	1.33	0	0	0.33	11.78
DenseNet169	0.28	4.34	0	0.10	1.18	15.18
DenseNet201	0	0.94	0.22	0	0.29	15.78
inceptionresnetv2	4.87	7.55	1.87	6.04	5.08	25.45
inceptionv3	3.13	5.91	2.35	4.25	3.91	19.17
mobilenet	3.22	2.11	6.72	14.16	6.55	15.61
mobilenetv2	0.36	1.52	0.03	1.04	0.74	11.51
nasnet	4.31	4.97	4.82	3.73	4.46	21.94
nasnetmobile	0.22	0.50	0	0.40	0.28	8.66
resnet101	0.02	0.29	0.02	0.21	0.14	12.38
resnet101v2	0	1.54	1.83	0.63	1.00	10.18
resnet152	0.85	1.37	0.85	1.02	1.02	11.36
resnet152v2	0	0.95	1.76	0.83	0.88	12.21
resnet50	1.13	0.39	0.82	0.62	0.74	7.89
resnet50v2	0.25	1.79	0.77	0.62	0.86	8.08
vgg16	0	0	0	0	0	4.18
vgg19	0	0.28	0	0	0.07	4.56
xception	5.78	9.29	3.32	8.71	6.77	18.62
AVG	1.36	2.50	1.41	2.35	1.91	13.03

ent models, backgrounds and camera views to those used during training. However, the adversarial texture remains highly transferable across different models. In our experiments, we found this effectiveness of our textures to be generally true. This is because our technique explicitly accounts for variations in scene parameters, which makes our textures robust to changes in scene conditions.

6. Conclusion

We explored adversarial texture generation with graphics pipeline in an end-to-end optimization paradigm, using differentiable rendering. We demonstrated highly effective adversarial textures for 3D models and established that our proposed optimization technique significantly outperform usage of target class images as the texture maps. We performed thorough experiments to demonstrate that our adversarial textures are able to fool 18 ImageNet models, even under considerable scene parameter variations.

7. Acknowledgements

The main author was recipient of an Australian Government Research Training Program (RTP) Scholarship at The University of Western Australia. This research was partially supported by ARC Discovery grant DP190102443. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00112090095. Dr. Naveed Akhtar is recipient of an Office of National Intelligence National Intelligence Postdoctoral Grant (project number NIPG-2021-001) funded by the Australian Government.

References

[1] Mahdieh Abbasi and Christian Gagné. Robustness to adversarial examples through an ensemble of specialists. *arXiv preprint arXiv:1702.06856*, 2017.

[2] Naveed Akhtar, Mohammad Jalwana, Mohammed Bennamoun, and Ajmal S Mian. Attack to fool and explain deep networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[3] Naveed Akhtar, Jian Liu, and Ajmal Mian. Defense against universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3389–3398, 2018.

[4] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *arXiv preprint arXiv:1801.00553*, 2018.

[5] Naveed Akhtar, Ajmal Mian, Navid Kardan, and Mubarak Shah. Advances in adversarial attacks and defenses in computer vision: A survey. *arXiv preprint arXiv:2108.00401*, 2021.

[6] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018.

[7] Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *Journal of Machine Learning Research*, 20(184):1–25, 2019.

[8] Tom B Brown et al. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[9] Guneet S Dhillon, Kamyar Azizzadenesheli, Zachary C Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442*, 2018.

[10] Ranjie Duan, Xingjun Ma, Yisen Wang, James Bailey, A Kai Qin, and Yun Yang. Adversarial camouflage: Hiding physical-world attacks with natural styles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1000–1008, 2020.

[11] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Brandon Tran, and Aleksander Madry. Adversarial robustness as a prior for learned representations. *arXiv preprint arXiv:1906.00945*, 2019.

[12] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *Journal of vision*, 16(12):326, 2016.

[13] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019.

[14] Ian J Goodfellow et al. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[16] Chengming Hu, Haolun Wu, and Kai Wang. Adversarial examples are not bugs, they are features, 2020. Submitted to NeurIPS 2019 Reproducibility Challenge.

[17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[18] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[19] Mohammad AAK Jalwana, Naveed Akhtar, Mohammed Bennamoun, and Ajmal Mian. Attack to explain deep representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9543–9552, 2020.

[20] Mohammad A. A. K. Jalwana, Naveed Akhtar, Mohammed Bennamoun, and Ajmal Mian. Attack to explain deep representation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9540–9549, 2020.

[21] Krishna Murthy Jatavallabhula, Edward Smith, Jean-Francois Lafleche, Clément Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, and Sanja Fidler. Kaolin: A pytorch library for accelerating 3d deep learning research. *ArXiv*, abs/1911.05063, 2019.

[22] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:1801.00553*, 2020.

[23] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. *arXiv preprint arXiv:1711.07566*, 2017.

[24] Ananya Kumar, S. M. Ali Eslami, Danilo J. Rezende, Marta Garnelo, Fabio Viola, Edward Lockhart, and Murray Shanahan. Consistent generative query network. *arXiv preprint arXiv:1807.02033*, 2019.

[25] Tzu-Mao Li. Differentiable visual computing. *arXiv preprint arXiv:1904.12228*, 2019.

[26] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.

[27] Matthew M. Loper and Michael J. Black. OpenDR: An approximate differentiable renderer. In *Computer Vision – ECCV 2014*, volume 8695 of *Lecture Notes in Computer Science*, pages 154–169. Springer International Publishing, Sept. 2014.

[28] David Lopez-Paz, Robert Nishihara, Soumith Chintala, Bernhard Schölkopf, and Léon Bottou. Discovering causal signals in images. *arXiv preprint arXiv:1605.08179*, 2017.

[29] Chunchuan Lyu, Kaizhu Huang, and Hai-Ning Liang. A unified gradient regularization family for adversarial examples. In *2015 IEEE International Conference on Data Mining*, pages 301–309. IEEE, 2015.

[30] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv preprint arXiv:1807.11164*, 2017.

[31] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[32] Aamir Mustafa, Salman Khan, Munawar Hayat, Roland Goecke, Jianbing Shen, and Ling Shao. Adversarial defense by restricting the hidden space of deep neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3385–3394, 2019.

[33] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38(6):1–17, 2019.

[34] Camilo Pestana, Naveed Akhtar, Wei Liu, David Glance, and Ajmal Mian. Adversarial attacks and defense on deep learning classification models using ycbcr color images. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9, 2021.

[35] Camilo Pestana, Wei Liu, David Glance, and Ajmal Mian. Defense-friendly images in adversarial attacks: Dataset and metrics for perturbation difficulty. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 556–565, 2021.

[36] Camilo Pestana, Wei Liu, David Glance, Robyn Owens, and Ajmal Mian. Assistive signals for deep neural network classifiers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 1221–1225, June 2021.

[37] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020.

[38] Hadi Salman, Andrew Ilyas, Logan Engstrom, Sai Vemprala, Aleksander Madry, and Ashish Kapoor. Unadversarial examples: Designing objects for robust vision. *arXiv preprint arXiv:2012.12235*, 2020.

[39] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *arXiv preprint arXiv:1801.04381*, 2019.

[40] Shibani Santurkar, Dimitris Tsipras, Brandon Tran, Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Computer vision with a single (robust) classifier. *CoRR*, abs/1906.09453, 2019.

[41] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *arXiv preprint arXiv:1904.12843*, 2019.

[42] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of supervised models through robust optimization. *Neurocomputing*, 307:195–204, 2018.

[43] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[44] Christian Szegedy et al. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[45] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[46] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, Rohit Pandey, Sean Fanello, Gordon Wetzstein, Jun-Yan Zhu, Christian Theobalt, Maneesh Agrawala, Eli Shechtman, Dan B Goldman, and Michael Zollhöfer. State of the art on neural rendering. *arXiv preprint arXiv:2004.03805*, 2020.

[47] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *arXiv preprint arXiv:2002.08347*, 2020.

[48] Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L Yuille. Improving transferability of adversarial examples with input diversity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2730–2739, 2019.

[49] Yunrui Yu, Xitong Gao, and Cheng-Zhong Xu. Lafeat: Piercing through adversarial defenses with latent features. *arXiv preprint arXiv:2104.09284*, 2021.

[50] Xiaohui Zeng, Chenxi Liu, Yu-Siang Wang, Weichao Qiu, Lingxi Xie, Yu-Wing Tai, Chi-Keung Tang, and Alan L Yuille. Adversarial attacks beyond the image space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4302–4311, 2019.

[51] Yang Zhang, Hassan Foroosh, Philip David, and Boqing Gong. CAMOU: Learning physical vehicle camouflages to adversarially attack detectors in the wild. In *International Conference on Learning Representations*, 2019.