Machine Learning in Adversarial Environments

by

Chaowei Xiao

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2020

Doctoral Committee:

Professor Mingyan Liu, Chair
Professor Atul Prakash
Professor Jason Corso
Assistant Professor David Fouhey
Assistant Professor Bo Li

Chaowei Xiao

xiaocw@umich.edu

ORCID iD: 0000-0002-7043-4926

Dedication

This manual is dedicated to all doctoral students at the University of Michigan's Horace H. Rackham School of Graduate Studies.

Acknowledgments

I would like to start with thanking my parents for unconditional supports. Thanks for their open-minded attitude towards me. I will never forget their sacrifice to ensure me to receive a quality education. Their selfless love and never-failing affectionate support have made me overcome the difficulties and pursue my academic dream.

I was very lucky to be a student advised by Professor Mingyan Liu. I am grateful for her providing me this valuable opportunity to work with her and giving me enough freedom, support, and encouragement for my research career. Beyond the tremendous and detailed advice that she has given on my professional career, she also taught me how to analyze the problem and how to build up the personal research taste. I am really grateful that she gave me enough time to study the fundamental knowledge and courses during my first two years instead of involving in a lot of research projects. I still remembered that the countless days and nights she has been spent on our research projects. Even she has become the chair of the ECE department, she still could sacrifice her lunchtime or spare time to talk with me. Additionally, she also offered me a lot of help in my personal life. I still remembered that she told me to sync with her frequently during the year when I was visiting at UC Berkeley because she hoped to confirm that I was safe and studying enjoyably instead of my research progress. She also provides me unconditional supports during the time of my job searching. She gave me enough time and freedom to prepare my job materials and spent a lot of time to help me revise the materials and rehearsal. Even after the interview process, she also helped me analyze the advantages and drawbacks of each offer and replied to my message during the mid-night. The work presented in this dissertation would not have been possible without her.

I am also extremely grateful to professor Bo Li. She provided me the precise and detailed advice and spent countless hours on my research and showed enough patient. I am so grateful to her recommendation of being a visiting student at UC Berkeley at Professor Dawn Songs group and also providing me the opportunity for academic services. Without her help and advice, the work presented in this dissertation would never come out.

I am also extremely thankful to professor Dawn Song. She provided me the opportunity to visit UC Berkeley and enough resources. I spent a really fruitful and wonderful time in her group.

I am also extremely grateful to professor Yunhao Liu, who was my advisor of my undergraduate university. I am thankful to his support and recommend me to join professor Mingyan Lius group. Never shall I forget the unconditional support and enough encouragement he provided. In the

meanwhile, I am also grateful to professor Lei Yang and Zheng Yang, who was my advisors of my undergraduate university as well.

I am also extremely grateful to professor Alfred Chen, professor Jia Deng, professor Jie Gao, professor Yang Liu, and professor Ning Zhang. They provided me a lot of support during my application for the academic position.

Special thanks to my close collaborators who made this thesis possible. I would like to thank Dawei Yang. He provided me a lot of help on 3D adversarial projects. I appreciate his many useful suggestions and patience during our collaboration. I would like to thank Warren He for his help on the projects of digital adversarial examples. I will never forget his inspiration and enthusiasm. I am looking forward to seeing the success of the mobile app he developed. I would like to thank Ruizhi Deng for his help on defense projects. He helped me conduct a huge amount of experiments. I would like to thank Yulong Cao for his help on LiDAR project. He taught the details of autonomous driving systems. He also provided me a lot of help when we first organized a workshop in CVPR.

Over the summers, I have had the fortune to intern with many researchers, including Denis Weng and Yu Chen at JD.COM, Ian Molloy and Taesung Lee at IBM Research, Hamid Palangi, Lei Zhang, Houdong Hu and Jianfeng Gao at Microsoft Research. I appreciate these opportunities provided by them.

I would also like to thank all of my other collaborators: Alfred Chen, Yulong Cao, Hongge Chen, Jia Deng, Ruizhi Deng, Tudor Dumitras, Kevin Eykholt, Ivan Evtimov, Earlence Fernanes,Kevin Fu, Jie Gao, Cho-Jui Hsieh, Warren He, Mo Li, Xiang-Yang Li, Kin Sum Liu, Yang Liu, Yunhao Liu, Honglake Lee, Ian Molloy, Xinlei Pan, Atul Prakash, Haonan Qiu, Amir Rahmati, Armin Sarabi, Liang Tong, Jian Tang, Yevgeniy Vorobeychik, Chenshu Wu, Gang Wang, Xinyu Xing, Zheng Yang, Lei Yang, Dawei Yang, Fisher Yu, Xinchen Yan, Ning Zhang, Ziyun Zhu, Huan Zhang, and Junyan Zhu. Thanks for your great efforts and help. Without your help and efforts, the work in this dissertation would never appear.

I would like to thank my labmates Xueru Zhang, Armin Sarabi, Yang Liu, Parinaz Naghizadeh, Mohammad Mahdi Khalili, Mehrdad Moharrami, Kun Jin, Chenlan Wang.

Finally, I would also like to thank my committee members Atul Prakash, Jason Corso, David Fouhey, and Bo Li. I am grateful to them for the valuable comments and suggestions for my dissertation.

TABLE OF CONTENTS

Dedication . . . . . **ii**

Acknowledgments . . . . . **iii**

List of Figures . . . . . **vii**

List of Tables . . . . . **xiii**

Abstract . . . . . **xv**

Chapter

1 Introduction **1**

2 Adversarial Example Crafting in the Digital Space **6**

2.1 Introduction 6
2.2 Generating Adversarial Examples using Adversarial Nets 8
2.2.1 AdvGAN Framework 8
2.3 A New Type of Adversarial Examples: Spatially Transformed Adversarial Examples 10
2.4 Experimental Results 12
2.4.1 Attack Effectiveness under Whitebox (*Semi-whitebox*) Setting 14
2.4.2 Visualizing the spatial transformation of stAdV 15
2.4.3 Human Perceptual Study 17
2.4.4 Attack Effectiveness Under Defenses 18
2.5 Conclusion 19

3 Adversarial Examples in the 3D Space **21**

3.1 Introduction 21
3.2 Problem Definition and Challenges 23
3.3 Methodology 23
3.3.1 Differentiable Rendering 24
3.3.2 Optimization Objective 25
3.4 Transferability to Black-Box Renderers 26
3.5 Experimental Results 27
3.5.1 Experimental Setup 28
3.5.2 **meshAdv** on Classification 29
3.5.3 **meshAdv** on Object Detection 33

3.5.4 Transferability to Black-Box Renderers . . . 34
3.6 Conclusion . . . 36

4 Adversarial Examples in the Physical World . . . 37

4.1 LiDAR-based Detection System . . . 39
4.2 Generating Adversarial Object Against LiDAR-based Detection . . . 41
4.2.1 Framework Overview . . . 41
4.2.2 Differentiable Renderer . . . 42
4.2.3 Differentiable Proxy Function . . . 43
4.2.4 Objective Functions . . . 45
4.2.5 Blackbox Attack . . . 46
4.3 Experiments . . . 46
4.3.1 Experimental Setup . . . 47
4.3.2 LiDARadv under Blackbox Settings . . . 47
4.3.3 LiDARadv with Different Adversarial Goals . . . 47
4.3.4 LiDARadv on Generating Robust Physical Adversarial Objects . . . 49
4.4 Conclusions . . . 52

5 Detecting Adversarial Examples by Using the Property of the Learning Model . . . 54

5.1 Spatial Consistency Based Method . . . 55
5.1.1 Spatial Context Analysis . . . 56
5.1.2 Patch Based Spatial Consistency . . . 57
5.2 Scale Consistency Analysis . . . 58
5.2.1 Scale Consistency Property . . . 59
5.3 Experimental Results . . . 60
5.3.1 Implementation Details . . . 60
5.3.2 Spatial Consistency Analysis . . . 61
5.3.3 Image Scale Analysis . . . 62
5.3.4 Adaptive Attack Evaluation . . . 63
5.3.5 Transferability Analysis . . . 64
5.4 Conclusions . . . 66

6 Detecting Adversarial Examples by Using the Property of the Data . . . 67

6.1 Adversarial frame identifier via temporal consistency: *AdvIT* . . . 69
6.1.1 Overview of Method . . . 70
6.2 Experimental Results . . . 75
6.2.1 Implementation Details . . . 75
6.2.2 Temporal Consistency Based Detection . . . 76
6.2.3 Analysis of Adaptive Attacks . . . 78
6.3 Discussion and Conclusion . . . 80

7 Conclusion and Future Directions . . . 82

Appendices . . . 85

Bibliography . . . 112

LIST OF FIGURES

FIGURE

2.1 Overview of AdvGAN 9
2.2 Generating adversarial examples with spatial transformation: the blue point denotes the coordinate of a pixel in the output adversarial image and the green point is its corresponding pixel in the input image. Red flow field represents the displacement from pixels in the adversarial image to pixels in the input image. 11
2.3 Adversarial examples generated from the same original image to different targets by AdvGAN and stAdv on MNIST. We use the format "model — attack method" to label the subcaption. On the diagonal, the original images are shown. 14
2.4 Comparison of adversarial examples generated by FGSM, C&W and stAdv. (Left: MNIST, right: CIFAR-10) The target class for MNIST is "0" and "air plane" for CIFAR-10. We generate adversarial examples by FGSM and C&W with perturbation bounded in terms of L_∞ as 0.3 on MNIST and 8 on CIFAR-10. 14
2.5 Adversarial examples generated by AdvGAN on CIFAR-10 and ImageNet. For the left image, the image from each class is perturbed to other different classes and the original images are shown on the diagonal. For the right image, images shown on the first column are the original images. The following columns are the corresponding adversarial examples which are classified as (from left to right) poodle, ambulance, basketball, and electric guitar. 15
2.6 Adversarial examples generated by stAdv against different models on CIFAR-10. The ground truth images are shown in the diagonal while the adversarial examples on each column are classified into the same class as the ground truth image within that column. 16
2.7 Flow visualization on MNIST. A digit "0" is misclassified as "2". 16
2.8 Flow visualization on CIFAR-10. An "airplane" image is misclassified as "bird". . . . 17
2.9 Flow visualization on ImageNet. (a): the original image, (b)-(c): images are misclassified into goldfish, dog and cat, respectively. Note that to display the flows more clearly, we fade out the color of the original image. 17

3.1 The pipeline of "adversarial mesh" generation by meshAdv. 22
3.2 Benign images (diagonal) and corresponding adversarial examples generated by meshAdv on *PASCAL3D+ renderings* tested on Inception-v3. Adversarial target classes are shown at the top. We show perturbation on (a) shape and (b) texture. 29

3.3 Benign images (diagonal) and corresponding adversarial examples generated by meshAdv on *PASCAL3D+ renderings* tested on Inception-v3. Adversarial target classes are shown at the top. We show perturbation on (a) shape and (b) texture. 30
3.4 (a) and (b) are visualization of shape based perturbation with respect to Figure 3.2(a). (c) is a close view of flow directions, and (d) is an example to compare the magnitude of perturbation with the magnitude of curvature. Warmer color indicates greater magnitude and vice versa. 31
3.5 "Adversarial meshes" generated by meshAdv in a synthetic indoor scene. (a) represents the benign rendered image and (b)-(e) represent the rendered images from "adversarial meshes" by manipulating the shape or texture. We use the format "adversarial target | perturbation type" to denote the victim object aiming to hide and the type of perturbation respectively. 33
3.6 "Adversarial meshes" generated by meshAdv for an outdoor photo. (a) and (c) show images rendered with pristine meshes as control experiments, while (b) and (d) contain "adversarial meshes" by manipulating the shape. We use the format " S/S_{adv} | target" to denote the benign/adversarial 3D meshes and the target to hide from the detector respectively. 34
3.7 Confusion matrices of targeted success rate for evaluating transferability of "adversarial meshes" on different classifiers. **Left**: DenseNet; **right**: Inception-v3. 35
3.8 Transferability of "adversarial meshes" against classifiers in unknown rendering environment. We estimate the camera viewpoint and lighting parameters using the differentiable renderer NMR, and apply the generated "adversarial mesh" to the photorealistic renderer Mitsuba. The "airliner" is misclassified to the target class "hammerhead" after rendered by Mitsuba. 36
3.9 Transferability of "adversarial meshes" against object detectors in unknown rendering environment. (b) (c) are controlled experiments. S^{adv} is generated using NMR (d), targeting to hide the leftmost chair (see red arrows), and the adversarial mesh is tested on Mit. (Mitsuba) (e). We use "S/S^{adv} | renderer" to denote whether the added object is adversarially optimized and the renderer that we aim to attack with transferability respectively. 36

4.1 Overview of LiDARadv. The first row shows the control experiment where a regular box is detected by the LiDAR-based detection system; row 2 shows the generated adversarial object with similar size cannot be detected. 38
4.2 Overview of LiDAR-based detection on AV. 39
4.3 The performance of trilinear approximator and tanh approximator. The format "$\phi(\cdot)''_{\mathbf{count}}$ represents the 2D count feature calculated by trilinear approximator Φ'; $M(\Phi'(X))_{\mathbf{obj}}$ represents visualization of the "objectiveness " metric in the output of model M using trilinear approximator with; $\Phi'(X)_{\mathbf{count}}$ - $\Phi(X)_{\mathbf{count}}$ represents the approximator 's error of Φ'. The same notation for tanh approximiator Φ'' 44
4.4 Adversarial meshes of different sizes can fool the detectors even with more LiDAR hits. We generate the object with LiDARadv and evolution-based method (Evo.). . . . 47
4.5 The adversarial mesh generated by LiDARadv is mis-detected as a "Pedestrian". . . . 48

4.6 The visualization of the adversarial object with different angles. In the benign frame (a), the system is able to detect the cube. When we replace the cube with our adversarial object, the system fails to detect the object at all three angles. We visualize the mesh along with the point clouds in a close-up view in (b), (c) and (d). 49
4.7 Our adversarial object can successfully attack the detection system, while placed at different positions. The red spheres mark the locations we place the adversarial object. 50
4.8 The optimized robust adversarial objects from 6 principal views and a particular view, compared with the original pristine object. 50
4.9 Results of physical attack. Our 3D-printed robust adversarial object by LiDARadv is not detected by the LiDAR-based detection system in a moving car. Row 1 shows the point cloud data collected by LiDAR sensor, and Row 2 presents the corresponding images captured by a dash camera. 51
4.10 Our physical experiment setting. We 3D-print the generated adversarial object at 1:1, and drive a car mounted with LiDAR and dashcams to collect the scanned point clouds and the reference videos. 52

5.1 Spatial consistency analysis for adversarial and benign instances in semantic segmentation. 55
5.2 Samples of benign and adversarial examples generated by Houdini on Cityscapes (targeting on Kitty/Pure) and BDD100K (targeting on Kitty/Scene). We select DRN as our target model here. Within each subfigure, the first column shows benign images and corresponding segmentation results, and the second and third columns show adversarial examples with different adversarial targets. 56
5.3 Heatmap of per-pixel self-entropy on Cityscapes dataset against DRN model. (a) and (b) show a benign image and its corresponding per-pixel self-entropy heatmap. (c)-(f) show the heatmaps of the adversarial examples generated by DAG and Houdini attacks targeting "Hello Kitty" (Kitty) and random pure color (Pure). 56
5.4 Examples of spatial consistency based method on adversarial examples generated by DAG and Houdini attacks targeting on Kitty and Pure. First column shows the original image and corresponding segmentation results. Column P_1 and P_2 show two randomly selected patches, while column O_1 and O_2 represent the segmentation results of the overlapping regions from these two patches, respectively. The mIOU between O_1 and O_2 are reported. It is clear that the segmentation results of the overlapping regions from two random patches are very different for adversarial images (low mIOU), but relatively consistent for benign instance (high mIOU). 57
5.5 Examples of images and corresponding segmentation results before/after image scaling on Cityscapes against DRN model. For each subfigure, the first column shows benign/adversarial image, while the later columns represent images after scaling by applying Gaussian kernel with std as 0.5, 3, and 5, respectively. (a) shows benign images before/after image scaling and the corresponding segmentation results; (b)-(e) present similar results for adversarial images generated by DAG and Houdini attacks targeting on Kitty and Pure. 60

5.6 Performance of adaptive attack. (a) shows adversarial image and corresponding segmentation result for adaptive attack against image scaling. The first two rows show benign images and the corresponding segmentation results; the last two rows show the adaptive adversarial images and corresponding segmentation results under different std of Gaussian kernel (0.5, 3, 5 for column 2-4). (b) and (c) show the performance of adaptive attack against spatial consistency based method with different K. (b) presents mIOU of overlapping regions for benign and adversarial images during along different iterations. (c) shows mIOU for overlapping regions of benign and adversarial instances at iteration 200. 63
5.7 Detection performance of spatial consistency based method against adaptive attack with different K on Cityscapes with DRN model. X-axis indicates the number of patches selected to perform the adaptive attack (0 means regular attack). Y-axis indicates the number of overlapping regions selected for during detection. 64
5.8 Transferability analysis: cell (i, j) shows the normalized mIoU value or pixel-wise attack success rate of adversarial examples generated against model j and evaluate on model i. Model A,B,C are DRN (DRN-D-22) with different initialization. We select "Hello Kitty" as target 65

6.1 Pipeline of the proposed temporal consistency based adversarial frame identifier: *AdvIT* . 68
6.2 Benign and adversarial frames generated on Cityscapes and Davis Challenge 17 dataset for video segmentation and human pose estimation respectively. 69
6.3 Benign and adversarial frames generated on MPII and UCF-101 for video object detection and action recognition respectively. 70
6.4 Benign and adversarial frames generated on MPII and UCF-101 for action recognition. 70
6.5 Heatmap of per-pixel cross-entropy. (a) and (b) show a benign frame and the corresponding per-pixel cross entropy between the prediction of its pseudo frames and itself. The rest show similar per-pixel cross entropy for adversarial frames with different targets. The labels indicate their adversarial targets. 73
6.6 Examples of consistency measurement based on *AdvIT* for various video tasks. The first column indicates previous Frames. The second column indicates the current frame and corresponding prediction result. The last column indicates a sampled pseudo frame and corresponding prediction. The consistency metric C shows quantitative results for different learning tasks. Note that higher C for segmentation and object detection means higher consistency, while lower C indicates more consistent for Human pose estimation since it is based on L_2 distance. 74

A.1 Samples of benign and adversarial examples. We use the format "attack method —attack model — dataset" to label the settings of each adversarial examples. Within each subfigure, the first column shows benign images and corresponding segmentation results, the second and third columns show adversarial examples with different adversarial targets (targeting on Kitty/Pure in (a),(c), (d) and on Kitty and Scene in (b),(d),(f)). 85
A.2 Attack results of additional targets on Cityscapes. The first column shows benign instance, while 2-4 columns show adversarial examples with target "ECCV 2018", "Remapping", and "Color strip", respectively. 86

A.3 Heatmap of per-pixel self-entropy. (a), (b), (g), (h), (m) and (n) show benign images and its corresponding per-pixel self-entropy heatmaps. We use the format "examples — attack model — dataset" to label them. For the rest, we use the format "attack method — target label — attack model — dataset" to label each subcaption. 90
A.4 Examples of images and corresponding segmentation results before/after image scaling. For each subfigure, the first column shows benign/adversarial images, while the following columns represent images after scaling by applying Gaussian kernel with std as 0.5, 3, and 5, respectively. (a),(f) and (k) show benign images before/after image scaling and the corresponding segmentation results and we use the format "example — attack model — dataset" to identify the corresponding model and dataset; (b)-(e), (g)-(j) and (l)-(o) present similar results for adversarial images and we use the format "attack method — target label — attack model — dataset" to label the settings of each image. 92
A.5 Examples of images and corresponding segmentation results for adaptive attack against image scaling. For each subfigure, the first column shows benign/adversarial images, while the following columns show images after scaling by applying Gaussian kernel with std as 0.5, 3, and 5, respectively. (a), (f), (k) and (p) show benign images before/after image scaling and the corresponding segmentation results and The format "example — attack model — dataset" uses to identify the corresponding model and dataset; (b)-(e), (g)-(j), (l)-(o) and (q)-(t) present similar results for adaptive adversarial images and we describe by the format "attack method — target label — attack model — dataset". 95
A.6 Detection performance of spatial consistency based method against adaptive attack with different K. We use the format "attack model — target label — dataset" to label the settings for each figure. X-axis indicates the number of patches selected to perform the adaptive attack (0 means regular attack). Y-axis indicates the number of overlapping regions selected during detection. We select the minimal mIOU from benign patches as our threshold on Cityscapes, and the one which guarantees accuracy as above 95% on benign images for BDD. 98
A.7 Transferability analysis on CityScapes dataset: cell (i, j) shows the normalized mIoU value or pixel-wise attack success rate of adversarial examples generated against model j and evaluate on model i. Model A,B,C have the same architecture (DRN-C-26 or DLA34UP) with different initialization. We use format "attack method | attack target | model " to denote the caption of each sub-figure. 100
A.8 Transferability analysis on BDD dataset. 102
A.9 Transferability visualization on CityScapes dataset. In each sub-figure, the first row presents the segmentation results of adversarial example on model A (targeted model) and model B. The second row shows the adversarial target and the ground truth. We use format "attack method | attack target | model " to denote the caption of each sub-figure. 103
A.10 Transferability visualization on BDD dataset. 105

A.11 Transferability analysis for classification models: cell (i, j) shows the attack success rate of the adversarial examples generated against Model j and evaluate on Model i under targeted attack setting. Model A,B,C are model with the same architecture (DRN-D-22, DRN-C-26 or DLA34UP) and different initialization. All the adversarial examples are generated using fast iterative gradient sign method. The caption of each sub-figure bear the "dataset | model". 106

LIST OF TABLES

TABLE

2.1 Comparison with the state-of-the-art attack methods. Run time is measured for generating 1,000 adversarial instances during test time. C&W represents the optimization based method, and Trans. denotes black-box attacks based on transferability. 8
2.2 Accuracy of different models on pristine data, and the attack success rate of adversarial examples generated against different models by AdvGAN on MNIST and CIFAR-10. . 13
2.3 Accuracy of different models on pristine data, and the attack success rate of adversarial examples generated against different models by stAdv on MNIST and CIFAR-10. . . 13
2.4 Attack success rate of adversarial examples generated by AdvGAN and stAdv under defenses on MNIST and CIFAR-10. 19

3.1 Attack success rate of meshAdv and average distance of generated perturbation for different models and different perturbation types. We choose rendering configurations in *PASCAL3D+ renderings* such that the models have 100% test accuracy on pristine meshes so as to confirm the adversarial effects. The average distance for shape based perturbation is computed using the 3D Laplacian loss from Equation 3.12. The average distance for texture based perturbation is the root-mean-squared error of face color change. 28
3.2 Targeted attack success rate for unseen camera views. We attack using 5, 10, or 15 views, and test with 20 unseen views in the same range. 32
3.3 Untargeted attack success rate against Mitsuba by transferring "adversarial meshes" generated by attacking a differentiable renderer targeting different classes. 35

4.1 Machine learning model input features extracted in the preprocessing phase. 40
4.2 Output metrics of the segmentation model. 40
4.3 Attack success rate of LiDARadv and evolution based method under different settings. 48
4.4 The attack success rate of the adversarial objects generating using LiDARadv, starting from different types of pristine meshes. The target labels are the other four labels different from the original predictions. 48
4.5 Robust Adversarial Object against different angles. The original confidence is x. Our success rate is 100%. (✓represents no object detected) 49
4.6 Robust Adversarial Object against different positions. The original object can be detected by Apollo. Our success rate is 100%. (✓represents no object detected) 50

4.7 Attack success rates of LiDARadv at different positions and orientations under both controlled and unseen settings. 51

5.1 Detection results (AUC) of image spatial (Spatial) and scale consistency (Scale) based methods on Cityscapes dataset. The number in parentheses of the Model shows the number of parameters for the target mode, and mIOU shows the performance of segmentation model on pristine data. We color all the AUC less than 80% with red. 62

6.1 Comparison of detection results (AUC) against different attacks for *AdvIT* and baseline methods. 77
6.2 Detection results (AUC) against *temporal continuity attack* 78
6.3 Detection results (AUC) of adaptive attacks and transferability analysis. 79
6.4 Detection overhead of *AdvIT* (in seconds). 80

A.1 Detection results (AUC) of image spatial (Spatial) and scale consistency (Scale) based methods on BDD dataset. The number in parentheses of the "Model" shows the number of parameters for the target mode, and "mIOU" shows the performance of segmentation model on pristine data. We color all the AUC less than 80% with red. 87
A.2 Detection results (AUC) of image spatial (Spatial) based method with random patch size on Cityscapes dataset. 88
A.3 Detection results (AUC) of image spatial (Spatial) based method with random patch size on BDD dataset. 88
A.4 Detection results (AUC) of spatial consistency (Spatial) based method on Cityscapes dataset for additional targets. 88
A.5 Detection results (AUC) of spatial consistency (Spatial) based method on BDD dataset for additional targets. 89

B.1 Detection results (AUC) of *AdvIT* against *independent frame attack* on various video tasks with different attack methods and targets. 109
B.2 Accuracy of the predicted pseudo-frames among different settings 110
B.3 Detection results (AUC) of differnt attack strength 110

ABSTRACT

Machine Learning, especially Deep Neural Nets (DNNs), has achieved great success in a variety of applications. Unlike classical algorithms that could be formally analyzed, there is less understanding of neural network-based learning algorithms. This lack of understanding through either formal methods or empirical observations results in potential vulnerabilities that could be exploited by adversaries. This also hinders the deployment and adoption of learning methods in security-critical systems.

Recent works have demonstrated that DNNs are vulnerable to carefully crafted adversarial perturbations. We refer to data instances with added adversarial perturbations as "adversarial examples". Such adversarial examples can mislead DNNs to produce adversary-selected results. Furthermore, it can cause a DNN system to misbehavior in unexpected and potentially dangerous ways. In this context, in this thesis, we focus on studying the security problem of current DNNs from the viewpoints of both *attack* and *defense*.

First, we explore the space of attacks against DNNs during the test time. We revisit the integrity of L_p regime and propose a new and rigorous *threat model* of adversarial examples. Based on this new threat model, we present the technique to generate adversarial examples in the digital space.

Second, we study the physical consequence of adversarial examples in the 3D and physical spaces. We first study the vulnerabilities of various vision systems by simulating the photo taken process by using the physical renderer. To further explore the physical consequence in the real world, we select the safety-critical application of autonomous driving as the target system and study the vulnerability of LiDAR-perceptual module. These studies show the potentially severe consequences of adversarial examples and raise awareness on its risks.

Last but not least, we develop solutions to defend against adversarial examples. We propose a consistency-check based method to detect adversarial examples by leveraging property of either the learning model or the data. We show two examples in segmentation task (leveraging learning model) and video data (leveraging the data), respectively.

CHAPTER 1

Introduction

Background Machine Learning, especially Deep Neural Networks (DNNs), has achieved great success in many applications [26, 48, 50, 70, 74]. The essence of most ML tasks is to approximate an unknown mapping from an input domain to an output domain given samples of input-output pairs. To obtain this mapping function, ML algorithms are proposed. These algorithms usually split the data into two sets: *training data* with both input and output (label) domain information and *test data* with only input domain information. They learn the mapping function using the training data and are then applied to the test data to assess their accuracy. These algorithms are developed based on a strong assumption that *the distribution of the training data is similar to the distribution of the test data*. However, this assumption introduces potential problems.

First of all, when we randomly sample data into a training set and a test set, it may introduce sampling bias (out-of-distribution data in the test set), which can be hard to eliminate and results in uncertain prediction results. More importantly, this assumption is built upon a benign environment free of adversarial manipulation in either the training data or the test data. In practice, there exist many incentives for data manipulation, some adversarial in nature, due to the wide deployment of DNNs system in a variety of safety-critical applications, such as spam detection and autonomous vehicles. For instance, spam detection is one of the first widely used application which employs machine learning to detect spam [34]. It did not take long for attackers to launch evasion attacks through content manipulating to evade detection and send spam [34]. In this thesis, we aim to study such security problems of modern DNNs by discovering and fixing identified vulnerabilities in adversarial environments where an attacker could perform adversarial manipulation.

Within this context, two types of security threats have been defined based on when adversarial manipulations are performed. An attack during the training stage is called a poisoning attack, while one during the test stage an evasion attack (or adversarial machine learning). In this thesis, we focus on the latter, the security threats on the test stage where parameters of a DNN are fixed.

The robustness of a targeted DNN. is typically evaluated/tested using generated input instances called adversarial examples. These adversarial examples are designed to lead to erroneous and undesirable output by the target DNN, but deemed harmless or ineffective when used on humans.

The problem of generating such adversarial examples is formalized as follows: Given a learned classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$ from an input domain $\mathcal{X}$ (e.g. image) to a set of classification outputs $\mathcal{Y}$ (e.g. label), an adversary aims to generate an adversarial example x_{adv} for an original instance $x \in \mathcal{X}$ with its ground truth label $y \in \mathcal{Y}$, so that the classifier predicts $f(x_{adv}) \neq y$ (untargeted attack) or $f(x_{adv}) = t$ (targeted attack) where t is the target class, while a typical human can easily (and correctly) recognize $f(x_{adv})$ as y.

Adversarial examples have been shown to subvert malware detection, fraud detection, or even potentially mislead autonomous navigation systems [38, 43, 95], and therefore pose security risks when applied to security-critical applications. A comprehensive study on adversarial examples is required to enable effective defense and to develop safe and reliable machine learning systems.

Motivation One important criterion for adversarial examples in the digital space is that the perturbed input should "look like" the original instances. Traditional attack strategies adopt $\mathcal{L}_2$ (or other $\mathcal{L}_p$) norm distance as a perceptual similarity metric to evaluate the distortion [44] [18, 41, 41, 91, 93, 95, 108, 112]. We will refer to this as the $\mathcal{L}_p$-based threat model. However, while the $\mathcal{L}_p$-norm distance measurement is a convenient source of adversarial perturbations, it is by no means a comprehensive description of all possible adversarial perturbations. It is not an ideal metric [58, 63] as $\mathcal{L}_2$ similarity is sensitive to lighting and viewpoint changes of a pictured object; or, an image can be shifted by one pixel, which leads to large $\mathcal{L}_\infty$ distance, but in both cases the translated image actually appears "the same" by human perception. Motivated by this, the first part of this thesis explores the space of adversarial examples in terms of generating new types of adversarial examples beyond $\mathcal{L}_p$-based threat model in the digital space. Note that, in this part, we mainly focus on the whitebox setting where the attacker know the perfect knowledge of the f including the network work parameters, to explore what a powerful adversary can do based on the Kerckhoffss principle [103] to better motivate defense methods. The works related to black-box settings where the attacker could not access to f including hard label [11, 21, 24] or soft label [3, 9, 22, 56, 114] attacks are not the main task of this thesis.

Adversarial examples produced using standard techniques often fail to fool classifiers in the physical world, when these examples were first captured over varying viewpoints and affected by natural phenomena such as lighting and camera noise [82, 83]. Additionally, attacks in the digital space based on adversarial examples through direct manipulation of pixels can be defended (relatively easily) by securing the camera, so that these generated images may not be realizable in practice. For this reason, there has been significant prior work on generating physically possible adversarial examples [5, 13, 38, 72], by altering the texture of a 3D surface, *i.e.* by applying adversarial printable 2D patches or painting patterns. Such attacks, however, are less suitable for textureless objects, because adding texture to an otherwise textureless surface may increase the

chance of it being detected and defended. Moreover, for texture-oblivious instruments such as Light Detection and Ranging (LiDAR) and Radio Detection and Ranging (RaDAR), texture-based perturbations will not make a difference and therefore are ineffective. Comprehensive studies including both texture and shape-based perturbations are thus needed. This motivates the second part of the thesis where we systematically study adversarial behaviors in the 3D and physical worlds.

A deep understanding of attack mechanisms and adversarial examples is only one side of the equation. Ultimately we need to build better and more robust learning systems that can defend themselves against these attacks. Many defenses have been proposed in the literature [18, 41, 91, 95, 108], only to be broken shortly after [6, 15], with the notable exception of adversarial training [41, 93, 112] and their variants [102, 143, 145]. In general, there are two types of defenses, detecting adversarial examples (detection) and making correct predictions on adversarial examples (adversarially robust classifier). The latter enables us to build a new (robust) machine learning model which could classify any inputs including adversarial examples and normal examples correctly. Adversarial training is an instance of the adversarially robust classifier, and is the most efficient algorithms in this category. However, adversarial training is very time-consuming because it requires generating adversarial examples during training and is only effective against small sets of adversarial examples that are generated in a similar way during training. Certificated robustness [35, 42, 90, 119, 123] is another instance of the adversarially robust classifier with a provable guarantees under certain ($\mathcal{L}_p$) threat model. Detecting adversarial examples belongs to the first category, and could be viewed as a binary classifier to distinguish adversarial from benign (normal) examples. Characterizing adversarial examples can enable us to identify valid input to DNNs. It can be embedded and deployed before the input layer of a DNN. Once an input is identified as adversarial, the system can reject it (or refuse to perform classification on it), thereby avoiding making an erroneous determination. The third and last part of the thesis is on improving the robustness of DNNs against adversarial input, by focusing on detection based methods that exploits properties inherent in either the learning models or the data, respectively.

Thesis Statement In this thesis, we focus on studying the vulnerabilities of DNNs in adversarial environments at test time, including both *attack* and *defense*.

Overview of the thesis On the *attack* side, we revisit the integrity of the $\mathcal{L}_p$-based threat model and propose a new rigorous threat model of adversarial examples. We argue that adversarial examples should be perceptually realistic examples that could fool the machine learning model without confusing human. We propose ways to generate adversarial examples based on the $\mathcal{L}_p$ and out of $\mathcal{L}_p$-based threat model respectively in *the digital space* in chapter 2.

We further study the physical consequence of adversarial examples *in the physical space*. We

systematically study the vulnerabilities of DNNs by simulating the photo-taking process with a physical renderer in Chapter 3. Based on this study, we then generate the physical adversarial examples against the real-world safety-critical system of autonomous vehicles in Chapter 4.

On the *defense* side, we propose a consistency-check based method to distinguish adversarial examples by leveraging the property of *the learning models* or *the data* in Chapter 6. We will show two examples in the segmentation task by using the property of the learning model) in Chapter 5 and video data by using the property of video data in Chapter 6 respectively.

Contributions of the thesis Our main contributions fall in two directions: *adversarial examples generation* and *characterizing adversarial examples*. The former can help us better understand the proprieties of adversarial behavior, which can further help us develop robust machine learning algorithms. In adversarial examples generation, our contributions are as follows.

- We introduce a new way to generate adversarial examples based on the $\mathcal{L}_p$-based threat model efficiently in chapter 2.
- We study the limitation of the commonly used $\mathcal{L}_p$-based threat model of adversarial examples and present a new one. Using this, we present a new type of adversarial examples, as opposed to manipulating the pixel values directly under the L_p regime. Our work provides a new direction in adversarial example generation and the design of corresponding defenses.
- We explore the adversarial examples in the 3D and physical world. In Chapter 3, we simulate the photo-taking process with a physical renderer and then generate adversarial examples to attack this process. We then propose an algorithm to physically attack the real-world safety-critical application, the autonomous driving system in Chapter 4. Moreover, instead of showing adversarial examples in the vision domain, we study the physical attack in the LiDAR-perceptual module of the autonomous driving system. We select the industry-level system, Baidu Apollo, as the target system. We show that adversarial examples also exist beyond vision components in the physical world .

On the defense front, we propose a consistency-check based method to distinguish adversarial examples by leveraging the properties of the learning models and the data. Specifically, we study the spatial consistency property of segmentation and observe that spatial consistency information can be leveraged to detect adversarial examples robustly even when a strong adaptive attacker has access to the model and detection strategy. Based on this observation, we propose a method to characterize adversarial examples based on spatial context information property of semantic segmentation in Chapter 5. Besides the explorations of the property of learning models, we further explore the property of the data and find that the temporal continuity property of the video could be used to

detects adversarial frames in video clips. Therefore, we develop a simple yet effective algorithm with 100% detection rate under different tasks: segmentation, human pose estimation, and object detection in Chapter 6. We show that this mechanism is robust against proposed strong adaptive adversaries as well.

Organization The remainder of this thesis is organized as follows. In Chapter 2, we will study the vulnerabilities of DNNs by generating adversarial examples. We will show what should be the threat model of adversarial examples and introduce the ways to generate them in digital space based on the published work [127, 128]. In Chapter 3 and Chapter 4, we go one step further to study the vulnerabilities of DNNs in the physical world. Specifically, we will describe our work [129] to simulate the photo-taken process to study the vulnerabilities of DNNs by using physical renderer in Chapter 3. Based on the studies in the simulation environment, we next introduce ways to generate adversarial examples physically to attack the real-world safety-critical application autonomous driving system in Chapter 4. Based on the studies of vulnerabilities of current DNNs, we study ways to fix the vulnerabilities of current DNNs in terms of detecting adversarial examples by leveraging the property of learning models [127] in Chapter 5 and the property of the data [131] in Chapter 6. Chapter 7 concludes and discusses potential future directions.

CHAPTER 2

Adversarial Example Crafting in the Digital Space

2.1 Introduction

This chapter describes how to craft adversarial examples in the digital spaces. With the discovery of the adversarial behavior by [107], different algorithms have been proposed for generating such adversarial examples, such as the gradient descent-based method (FGSM) [41, 93] and optimization-based methods (CW) [16, 80].

The current attack algorithms [16, 80] rely on optimization schemes with simple pixel space metrics, such as L_∞ distance from a benign image, to encourage visual realism. Such optimization-based strategy limits the generation speed due to the requirements of the multiple forward and backward passes. To generate perceptually realistic adversarial examples efficiently, we propose to train a feed-forward network to generate perturbations such that the resulting examples must be realistic according to a discriminator network. We apply generative adversarial networks (GANs) [40] to produce adversarial examples. As conditional GANs are capable of producing high-quality images [58], we apply a similar paradigm to produce perceptually realistic adversarial instances. We name this attack method AdvGAN. Note that in the previous white-box attacks, such as FGSM and optimization methods, the adversary needs to have white-box access to the architecture and parameters of the model all the time. However, by deploying AdvGAN, once the feed-forward network is trained, it can instantly produce adversarial perturbations for any input instances without requiring access to the model itself anymore. We name this attack setting *Semi-whitebox* .

In general, as shown in Table 2.1, in terms of computation efficiency AdvGAN performs much faster than others even including the efficient FGSM, although AdvGAN needs extra training time to train the generator. In addition, although the perturbations generated by AdvGAN are constrained with $\mathcal{L}_p$-distance metric, as we leverage the GAN strategy, it could provide the new realism term (GAN loss) to benefit these adversarial instances in appearing closer to real instances compared to other attack strategies (CW &. FGSM). It could potentially help to explore the $\mathcal{L}_p$-based adversarial subspace more thoroughly. To verify it, we also apply the state-of-the-art adversarial training based

defense methods [41, 93, 112] trained on $\mathcal{L}_p$-based adversarial examples to defend against the adversarial examples generated by AdvGAN. Our results show that adversarial examples generated by AdvGAN can achieve a higher attack success rate which potentially verifies our guess.

Before moving towards more advanced methods, we take a step back and review previous methods. For the previous method, a $\mathcal{L}_p$ norm distance metric is used as a perceptual similarity metric to evaluate the distortion of adversarial examples. We name it $\mathcal{L}_p$-based threat model for convenience. This distance metric enables the adversarial examples should look like the original instances. However, while the $\mathcal{L}_p$-norm distance measurement is a convenient source of adversarial perturbations, it is by no means a comprehensive description of all possible adversarial perturbations. Moreover, $\mathcal{L}_p$-norm distance metric is not an ideally perceptual metric [58, 63], as $\mathcal{L}_2$ distance metric is sensitive to lighting and viewpoint change of a pictured object and $\mathcal{L}_\infty$ distance metric is sensitive to position shifting. Therefore, in this chapter, we give a new rigorous threat model of adversarial examples. We argue that adversarial examples should be the perceptually realistic examples which could fool the machine learning model without confusing human. Based on this new threat model, another goal in this chapter is to look for other types of adversarial examples beyond the $\mathcal{L}_p$-based threat model.

Based on this new threat model, we propose a new method named stAdv which explores the new adversarial space to generate adversarial examples. Different from previous adversarial examples (e.g. AdvGAN, FGSM, CW) with the additive perturbations, stAdv creates perceptually realistic examples by changing the positions of pixels instead of directly manipulating existing pixel values, which has been shown to better preserve the identity and structure of the original image [149]. In order to verify whether the adversarial examples generated by stAdv is a new type of adversarial examples beyond the $\mathcal{L}_p$-based threat model, we evaluate them by using the same adversarial training based defense methods which are trained on $\mathcal{L}_p$-based adversarial examples. The results show that previous adversarial training based defense method may appear less effective against stAdv because the spatially transformed adversarial examples are generated through a rather different principle, whereby what is being minimized is the local geometric distortion rather than the $\mathcal{L}_p$ pixel error between the adversarial and original instances. This fact convinces that the adversarial examples generated by stAdv have never been seen before.

stAdv broadens attack generation beyond L_p-norm regime. It opens up a new challenge on how to defend against such attacks, as well as other attacks that are not based on direct pixel value manipulation. We also visualize the spatial deformation generated by stAdv; it is seen to be locally smooth and virtually imperceptible to the human eye.

Our contributions in this chapter are summarized as follows:

- We proposed AdvGAN to train a conditional adversarial network to directly produce $\mathcal{L}_p$-based adversarial examples , which are both perceptually realistic and achieve state-of-the-art attack

success rate against different target models.

- We use state-of-the-art defense methods to defend against adversarial examples and show that AdvGAN achieves a higher attack success rate under current defenses.
- We explore the new space which containing the adversarial examples and propose stAdv to generate a new type of adversarial examples based on spatial transformation instead of direct manipulation of the pixel values.
- We provide visualizations of optimized transformations and show that such geometric changes are small and locally smooth, leading to high perceptual quality.
- We empirically show that, compared to other attacks, adversarial examples generated by stAdv are more difficult to detect with current defense systems.

2.2 Generating Adversarial Examples using Adversarial Nets

To generate perceptually realistic adversarial examples efficiently, we propose to train a feed-forward network to generate perturbations such that the resulting example must be realistic according to a discriminator network. We apply generative adversarial networks (GANs) [40] to produce adversarial examples. We named this algorithm AdvGAN. Once the feed-forward network is trained, it can instantly produce adversarial perturbations for any input instances without requiring access to the model itself anymore. Table 2.1 shows the advantages of AdvGAN compared to others.

Table 2.1: Comparison with the state-of-the-art attack methods. Run time is measured for generating 1,000 adversarial instances during test time. C&W represents the optimization based method, and Trans. denotes black-box attacks based on transferability.

	FGSM	C&W	Trans.	AdvGAN
Run time	0.06s	>3h	-	¡0.01s
target Attack	✓	✓	Ens.	✓

2.2.1 AdvGAN Framework

Figure 2.1 illustrates the overall architecture of AdvGAN, which mainly consists of three parts: a generator $\mathcal{G}$, a discriminator $\mathcal{D}$, and the target neural network f. Here the generator $\mathcal{G}$ takes the original instance x as its input and generates a perturbation $\mathcal{G}(x)$. Then $x + \mathcal{G}(x)$ will be sent to the discriminator $\mathcal{D}$, which is used to distinguish the generated data and the original instance x. The goal of $\mathcal{D}$ is to encourage that the generated instance is indistinguishable with the data from its original class. To fulfill the goal of fooling a learning model, we first perform the white-box

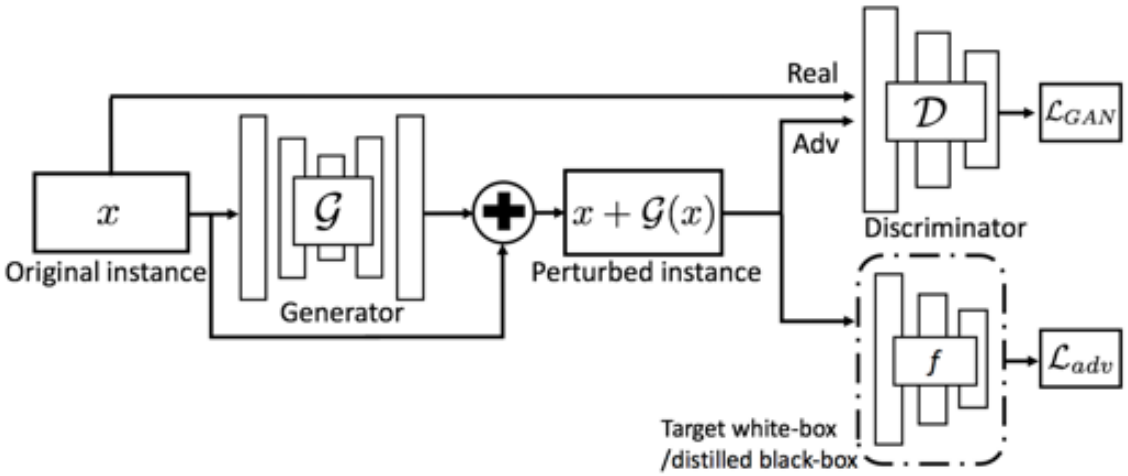


Figure 2.1: Overview of AdvGAN

attack, where the target model is f in this case. f takes $x + \mathcal{G}(x)$ as its input and outputs its loss $\mathcal{L}_{adv}$, which represents the distance between the prediction and the target class t (target attack), or the opposite of the distance between the prediction and the ground truth class (untarget attack).

The GAN loss [40] can be written as: [1]

$$\mathcal{L}_{\text{GAN}} = \mathbb{E}_x \log \mathcal{D}(x) + \mathbb{E}_x \log(1 - \mathcal{D}(x + \mathcal{G}(x))). \tag{2.1}$$

Here, the discriminator $\mathcal{D}$ aims to distinguish the perturbed data $x + \mathcal{G}(x)$ from the original data x.[2] Note that the real data is sampled from the true class, so as to encourage that the generated instances are close to data from the original class.

The loss for fooling the target model f in a target attack is:

$$\mathcal{L}_{\text{adv}}^{f} = \mathbb{E}_x \ell_f(x + \mathcal{G}(x), t), \tag{2.2}$$

where t is the target class and ℓ_f denotes the loss function (e.g., cross-entropy loss) used to train the original model f. The $\mathcal{L}_{adv}^{f}$ loss encourages the perturbed image to be misclassified as target class t. Here we can also perform the untarget attack by maximizing the distance between the prediction and the ground truth, but we will focus on the target attack in the rest of this section.

To generate adversarial examples under the $\mathcal{L}_p$ threat model, we bound the magnitude of the perturbation, which is a common practice in prior work [8, 16, 80]. For instance, here, we add a soft hinge loss on the L_2 norm as

$$\mathcal{L}_{\text{hinge}} = \mathbb{E}_x \max(0, \|\mathcal{G}(x)\|_2 - c), \tag{2.3}$$

[1] For simplicity, we denote the $\mathbb{E}_x \equiv \mathbb{E}_{x \sim \mathcal{P}_{\text{data}}(x)}$

[2] Note that we only use the generator to produce the perturbation $\mathcal{G}(x)$.

where c denotes a user-specified bound. This can also stabilize the GAN's training, as shown in [58].

Finally, our full objective can be expressed as

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{\text{adv}}^{f} + \mathcal{L}_{perceptual} \\ &= \mathcal{L}_{\text{adv}}^{f} + \alpha\mathcal{L}_{\text{GAN}} + \beta\mathcal{L}_{\text{hinge}},\end{aligned} \tag{2.4}$$

where α and β control the relative importance of each objective. The perceptual loss consists of GAN loss ($\mathcal{L}_{\text{GAN}}$ and hinge loss $\mathcal{L}_{hinge}$. Note that $\mathcal{L}_{\text{GAN}}$ here is used to encourage the perturbed data to appear similar to the original data x, while $\mathcal{L}_{\text{adv}}^{f}$ is leveraged to generate adversarial examples, optimizing for the high attack success rate. We obtain our $\mathcal{G}$ and $\mathcal{D}$ by solving the minmax game $\arg\min_{\mathcal{G}}\max_{\mathcal{D}}\mathcal{L}$

2.3 A New Type of Adversarial Examples: Spatially Transformed Adversarial Examples

In computer vision and graphics literature, two main aspects determine the appearance of a pictured object [110]: (1) the *lighting and material*, which determine the brightness of a point as a function of illumination and object material properties, and (2) the *geometry*, which determines where the projection of a point will be located in the scene. Most previous adversarial attacks [41] build on changing the *lighting and material* aspect, while assuming the underlying geometry stays the same during the adversarial perturbation generation process. Moreover, in the literature, adversarial examples are described as datapoints which are added by imperceptible perturbations bounded by L_p-norm regime to existing datapoints. It is a pretty limited set of manipulation function and the bounded L_p-norm regime is not necessarily the best choice of imperceptibility. For instance, L_2-norm is sensitive to lighting and viewpoint changes of a pictured object; shifting pixels will lead to large L_{inf} distance, while the translated image appears the same to human perception.

Therefore, our research sought to broaden attack generation beyond $\mathcal{L}_p$-norm regime. We argued that *the threat model of adversarial examples should be perceptually realistic inputs which could be correctly recognized by humans but mislead machine learning models.* Inspired by this threat model, we proposed stAdv to create perceptually realistic examples to fool machine learning models by changing pixel positions instead of directly manipulating existing pixel values. We call this method stAdv. Figure 2.2 shows the pipeline of stAdv.

In the following paragraphs, we introduce our geometric image formation model and then describe the objective function for generating spatially transformed adversarial examples.

Spatial transformation We use $x_{adv}^{(i)}$ to denote the pixel value of the i-th pixel and 2D coordinate $(u_{\text{adv}}^{(i)}, v_{\text{adv}}^{(i)})$ to denote its location in the adversarial image x_{adv}. We assume that $x_{adv}^{(i)}$ is transformed

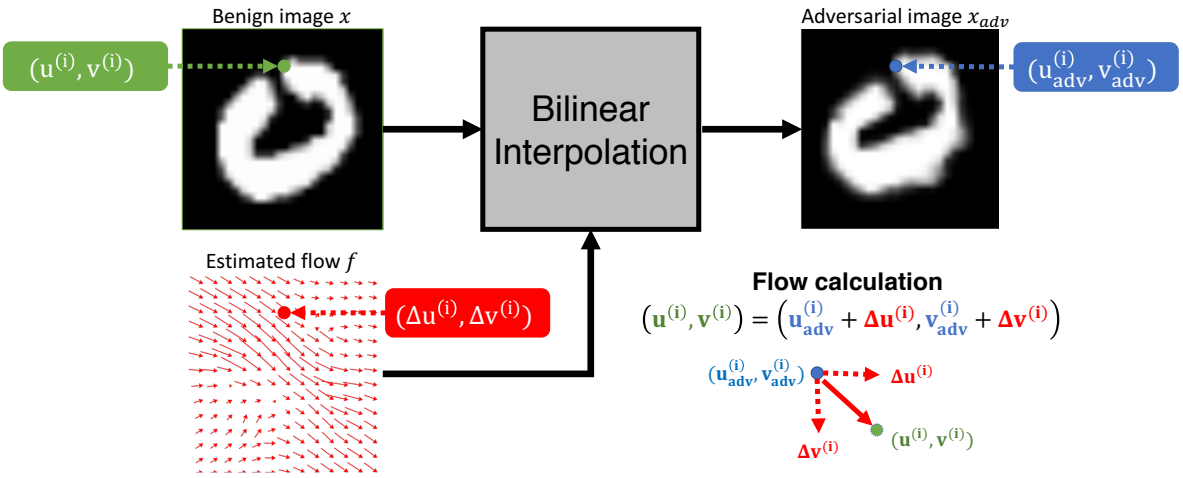


Figure 2.2: Generating adversarial examples with spatial transformation: the blue point denotes the coordinate of a pixel in the output adversarial image and the green point is its corresponding pixel in the input image. Red flow field represents the displacement from pixels in the adversarial image to pixels in the input image.

from the pixel $\mathbf{x}^{(i)}$ from the original image. We use the per-pixel flow (displacement) field $\mho$ to synthesize the adversarial image x_{adv} using pixels from the input $\mathbf{x}$. For the i-th pixel within x_{adv} at the pixel location $(u^{(i)}_{\text{adv}}, v^{(i)}_{\text{adv}})$, we optimize the amount of displacement in each image dimension, with the pair denoted by the *flow vector* $f_i := (\Delta u^{(i)}, \Delta v^{(i)})$. Note that the flow vector $\mho_i$ goes from a pixel $x^{(i)}_{adv}$ in the adversarial image to its corresponding pixel $\mathbf{x}^{(i)}$ in the input image. Thus, the location of its corresponding pixel $\mathbf{x}^{(i)}$ can be derived as $(u^{(i)}, v^{(i)}) = (u^{(i)}_{\text{adv}} + \Delta u^{(i)}, v^{(i)}_{\text{adv}} + \Delta v^{(i)})$. As the $(u^{(i)}, v^{(i)})$ can be fractional numbers and does not necessarily lie on the integer image grid, we use the differentiable bilinear interpolation [59] to transform the input image with the flow field. We calculate $x^{(i)}_{adv}$ as:

$$x^{(i)}_{adv} = \sum_{q \in \mathcal{N}(u^{(i)}, v^{(i)})} \mathbf{x}^{(q)} (1 - |u^{(i)} - u^{(q)}|)(1 - |v^{(i)} - v^{(q)}|), \tag{2.5}$$

where $\mathcal{N}(u^{(i)}, v^{(i)})$ are the indices of the 4-pixel neighbors at the location $(u^{(i)}, v^{(i)})$ (top-left, top-right, bottom-left, bottom-right). We can obtain the adversarial image x_{adv} by calculating Equation 2.5 for every pixel $x^{(i)}_{adv}$. Note that x_{adv} is differentiable with respect to the flow field f [59, 149]. The estimated flow field essentially captures the amount of spatial transformation required to fool the classifier.

Objective function Most of the previous methods constrain the added perturbation to be small regarding a $\mathcal{L}_p$ metric. Here instead of imposing the $\mathcal{L}_p$ norm on pixel space, we introduce a new regularization loss $\mathcal{L}_{flow}$ on the local distortion $\mho$, producing higher perceptual quality for

adversarial examples. Therefore, the goal of the attack is to generate adversarial examples which can mislead the classifier as well as minimizing the local distortion introduced by the flow field $\mho$.

Formally, we could put all of above to our universal objective function. Given a benign instance $\mathbf{x}$, we obtain the flow field $\mho$ by minimizing the following objective:

$$\mho^* = \underset{\mho}{\operatorname{argmin}} \quad \mathcal{L}_{adv}(x, \mho) + \tau \mathcal{L}_{flow}(\mho), \tag{2.6}$$

where $\mathcal{L}_{adv}$ encourages the generated adversarial examples to be misclassified by the target classifier. L_{flow} ensures that the spatial transformation distance is minimized to preserve high perceptual quality, and τ balances these two losses.

The goal of $\mathcal{L}_{adv}$ is to guarantee the target attack $g(x_{adv}) = t$ where t is the target class, different from the ground truth label y. Recall that we transform the input image $\mathbf{x}$ to x_{adv} with the flow field $\mho$ (Equation 2.5). In practice, directly enforcing $f(x_{adv}) = t$ during optimization is highly non-linear, we adopt the objective function suggested in [16].

$$\mathcal{L}_{adv}(x, \mho) = \max(\max_{i \neq t} f(x_{adv})_i - f(x_{adv})_t, -\kappa), \tag{2.7}$$

where $f_i(x)$ represents the i-th element of the output (logit) of model f, and κ is used to control the attack confidence level.

To compute $\mathcal{L}_{flow}$, we calculate the sum of spatial movement distance for any two adjacent pixels. Given an arbitrary pixel p and its neighbors $q \in \mathcal{N}(p)$, we enforce the locally smooth spatial transformation perturbation $\mathcal{L}_{flow}$ based on the total variation [100]:

$$\mathcal{L}_{flow}(\mho) = \sum_{p}^{\textit{all pixels}} \sum_{q \in \mathcal{N}(p)} \sqrt{||\Delta u^{(p)} - \Delta u^{(q)}||_2^2 + ||\Delta v^{(p)} - \Delta v^{(q)}||_2^2}. \tag{2.8}$$

Intuitively, minimizing the spatial transformation can help ensure the high perceptual quality for stAdv, since adjacent pixels tend to move towards close direction and distance. We solve the above optimization with L-BFGS solver [79].

2.4 Experimental Results

In this section, we first evaluate AdvGAN under *Semi-whitebox* and stAdv under whitebox settings on MNIST [73] , CIFAR-10 [71] and ImageNet [70]. We apply AdvGAN and stAdv to generate adversarial examples on different target models and evaluate the attack success rate for them under state-of-the-art defenses to show that our methods can achieve higher attack success rates compared to other existing attack strategies (FGSM and CW). We generate all adversarial examples for different attack methods based on the under L_∞ bound of 0.3 on MNIST and 8 on

Table 2.2: Accuracy of different models on pristine data, and the attack success rate of adversarial examples generated against different models by AdvGAN on MNIST and CIFAR-10.

	MNIST			CIFAR-10	
Model	A	B	C	ResNet-32	Wide ResNet-34
Accuracy	98.97%	99.17%	99.09%	92.41%	95.01%
Attack Success Rate	97.9%	97.1%	98.3%	94.71%	99.30%

Table 2.3: Accuracy of different models on pristine data, and the attack success rate of adversarial examples generated against different models by stAdv on MNIST and CIFAR-10.

	MNIST			CIFAR-10	
Model	A	B	C	ResNet-32	Wide ResNet-34
Accuracy	98.58%	98.94%	99.11%	93.16%	95.82%
Attack Success Rate	99.95%	99.98%	100.00%	99.56%	98.84%

CIFAR-10 and ImageNet, for a fair comparison.

For MNIST, in all of our experiments, Models A and B are used in [113], which represent different architectures. For CIFAR-10, we select ResNet-32 and Wide ResNet-34 [48, 141] for our experiments. Specifically, we use a 32-layer ResNet implemented in TensorFlow[3] and Wide ResNet derived from the variant of "w32-10 wide."[4] We show the classification accuracy of pristine MNIST and CIFAR-10 test data in Table 2.2 and Table 2.3 [5].

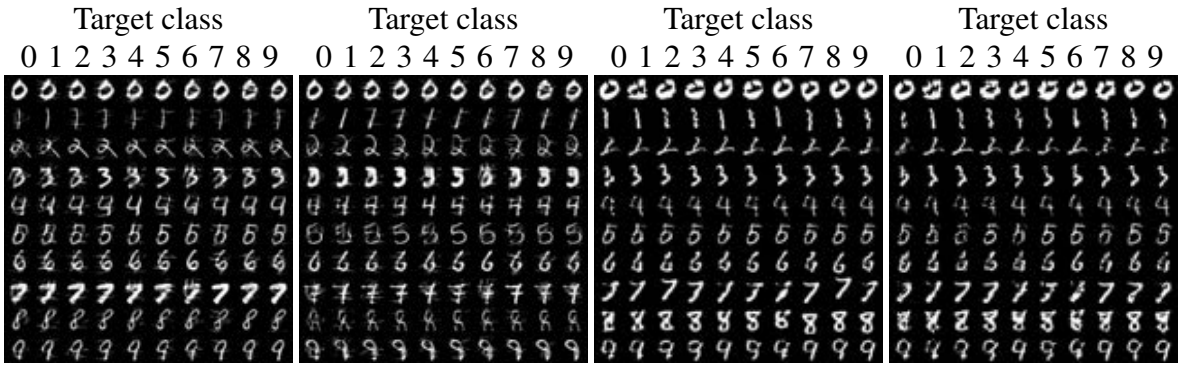
Implementation Details of AdvGAN We adopt a similar architecture from image-to-image translation literature [58, 151]. In particular, we use the architecture of generator $\mathcal{G}$ from [63], and our discriminator $\mathcal{D}$'s architecture is similar to the model used in [16] for MNIST and ResNet-32 for CIFAR-10. We apply the loss in [18] as our loss $\mathcal{L}_{adv}^{f} = \max(\max_{i \neq t} f(x_{adv})_i - f(x_{adv})_t, -\kappa)$, where t is the target class, and f represents the target network in the *Semi-whitebox* setting. We set the confidence $\kappa = 0$ for both CW and AdvGAN. We use Adam as our solver [67], with a batch size of 128 and a learning rate of 0.001. For GANs training, we use the least squares objective proposed by LSGAN [86], as it has been shown to produce better results with more stable training.

Implementation Details of stAdv We use the same target models of AdvGAN with different random seed for stAdv. We set τ as 0.05 for all our experiments. We use confidence $\kappa = 0$ for both CW and stAdv for a fair comparison.

[3] https://github.com/tensorflow/models/blob/master/research/ResNet/ResNet_model.py

[4] https://github.com/MadryLab/cifar10_challenge/blob/master/model.py

[5] Note that, we use the same architecture of AdvGAN with different random seed to train the models of stAdv.



(a) A — AdvGAN (b) B — AdvGAN (c) A — stAdv (d) B — stAdv

Figure 2.3: Adversarial examples generated from the same original image to different targets by AdvGAN and stAdv on MNIST. We use the format "model — attack method" to label the subcaption. On the diagonal, the original images are shown.

2.4.1 Attack Effectiveness under Whitebox (*Semi-whitebox*) Setting

We apply AdvGAN and stAdv to perform *Semi-whitebox* and whitebox attack against each model on MNIST dataset respectively. From the performance shown in Table 2.2 and Table 2.3, we can see that both AdvGAN and stAdv are able to generate adversarial instances to attack all models with high attack success rates.

Figure 2.4: Comparison of adversarial examples generated by FGSM, C&W and stAdv. (Left: MNIST, right: CIFAR-10) The target class for MNIST is "0" and "air plane" for CIFAR-10. We generate adversarial examples by FGSM and C&W with perturbation bounded in terms of L_∞ as 0.3 on MNIST and 8 on CIFAR-10.

We also visualize the adversarial examples generated from the same original instance x and predicted as the different target classes in Figure 2.3, Figure 2.5 and Figure 2.6. We can see that all of the generated adversarial examples for different models appear close to the ground truth/pristine images (lying on the diagonal of the matrix) and can be successfully misclassified as the target class shown on the top. If we compare the adversarial examples generated by AdvGAN and stAdv, we could find that adversarial examples generated by stAdv slightly change the shape of the digits but still look visually realistic. Figure 2.4 further visualize the adversarial examples generated by stAdv with the other $\mathcal{L}_p$ based adversarial examples (FGSM and CW). It shows the difference adversarial space explored by stAdv.

(a) CIFAR (b) ImageNet

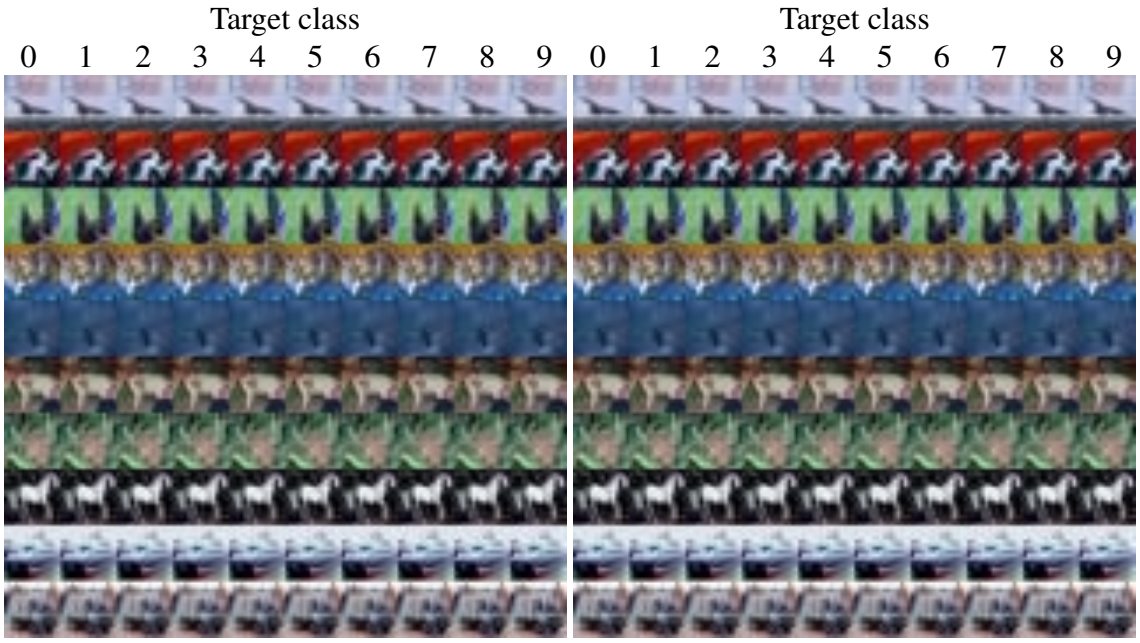
Figure 2.5: Adversarial examples generated by AdvGAN on CIFAR-10 and ImageNet. For the left image, the image from each class is perturbed to other different classes and the original images are shown on the diagonal. For the right image, images shown on the first column are the original images. The following columns are the corresponding adversarial examples which are classified as (from left to right) poodle, ambulance, basketball, and electric guitar.

One of the potentially limitations of the GAN-based algorithms is that the ability to generate high resolution images. To evaluate AdvGAN's ability to generate high resolution adversarial examples, we also apply AdvGAN to generate adversarial examples on the ImageNet as shown in the right of Figure 2.5 with L_∞ bound as 8. The added perturbation is unnoticeable while all the adversarial instances are misclassified into other target classes with high confidence.

For AdvGAN, we also analyze the influence of different loss terms on MNIST. Under the same bounded perturbations (0.3), if we replace the full loss function in equation 2.4 with $\mathcal{L} = ||\mathcal{G}(x)||_2 + \mathcal{L}_{\text{adv}}^f$, which is similar to the objective used in [7], the attack success rate becomes 86.2%. If we replace the loss function with $\mathcal{L} = \mathcal{L}_{\text{hinge}} + \mathcal{L}_{\text{adv}}^f$, the attack success rate is 91.1%, compared to that of AdvGAN, 98.3%. The potential reason is that our GAN-based design could help explore the adversarial space more thoroughly.

2.4.2 Visualizing the spatial transformation of stAdV

To better understand the spatial transformation applied to the original images, we visualize the optimized transformation flow for different datasets, respectively. Figure 2.7 visualizes a transformation on an MNIST instance, where the digit "0" is misclassified as "2." We can see that



(a) wide ResNet34 (b) ResNet32

Figure 2.6: Adversarial examples generated by stAdv against different models on CIFAR-10. The ground truth images are shown in the diagonal while the adversarial examples on each column are classified into the same class as the ground truth image within that column.

the adjacent flows move in a similar direction in order to generate smooth results. The flows are more focused on the edge of the digit and sometimes these flows move in different directions along the edge, which implies that the object boundary plays an important role in our stAdv optimization. Figure 2.8 illustrates a similar visualization on CIFAR-10. It shows that the optimized flows often focus on the area of the main object, such as the airplane. We also observe that the magnitude of flows near the edge are usually larger, which similarly indicates the importance of edges for misleading the classifiers. This observation confirms the observation that when DNNs extract edge information in the earlier layers for visual recognition tasks [117]. In addition, we visualize the

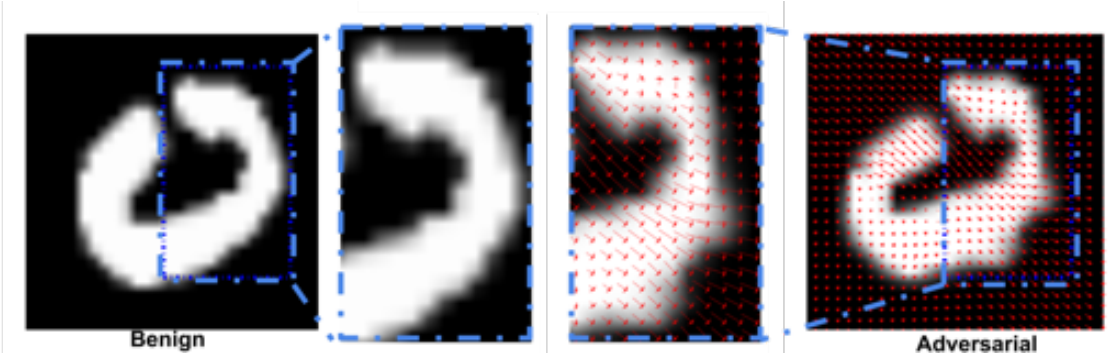


Figure 2.7: Flow visualization on MNIST. A digit “0” is misclassified as “2”.

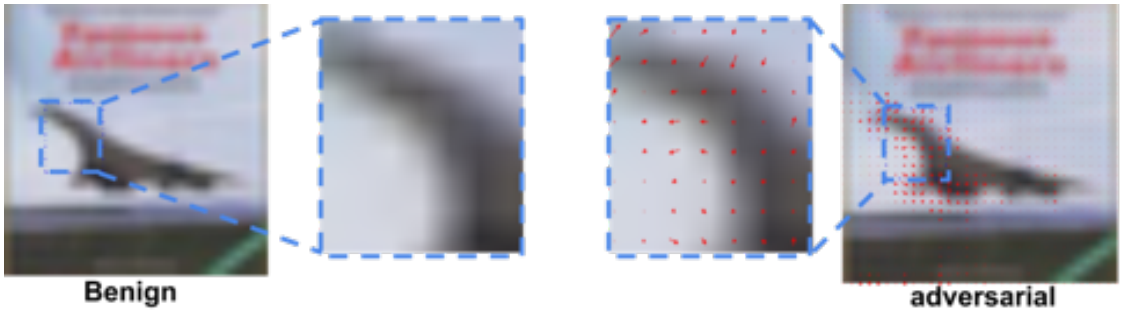


Figure 2.8: Flow visualization on CIFAR-10. An "airplane" image is misclassified as "bird".

(a) mountain bike (b) goldfish (c) Maltese dog (d) tabby cat

Figure 2.9: Flow visualization on ImageNet. (a): the original image, (b)-(c): images are misclassified into goldfish, dog and cat, respectively. Note that to display the flows more clearly, we fade out the color of the original image.

similar flow for the ImageNet dataset [32] in Figure 2.9. The top-1 label of the original image in Figure 2.9 (a) is "mountain bike". Figure 2.9 (b)-(d) show target adversarial examples generated by stAdv, which have target classes "goldfish," "Maltese dog," and "tabby cat," respectively, and which are predicted as such as the top-1 class. An interesting observation is that, although there are other objects within the image, nearly 90% of the spatial transformation flows tend to focus on the target object bike. Different target class corresponds to different directions for these flows, which still fall into the similar area.

2.4.3 Human Perceptual Study

To quantify the perceptual realism of the adversarial examples generated by AdvGAN and stAdv, we perform a user study with human participants on Amazon Mechanical Turk (AMT). We follow the same perceptual study protocol used in prior image synthesis work [58, 146]. In our study, the participants are asked to choose the more *visually realistic* image between an adversarial example generated by AdvGAN or stAdv and its original image. During each trial, these two images appear side-by-side for 2 seconds. After the images disappear, our participants are given unlimited time to make their decision. To avoid labeling bias, we allow each user to conduct at most 50 trails. For each pair of an original image and its adversarial example, we collect about 5 annotations from

different users.

Examples generated by AdvGAN and stAdv were chosen as the more realistic in $49.4\% \pm 1.96\%$ and $47.01\% \pm 1.96\%$ of the trails (perfectly realistic results would achieve 50%) respectively. This indicates that our adversarial examples are almost indistinguishable from natural images.

2.4.4 Attack Effectiveness Under Defenses

Facing different types of attack strategies, various defenses have been provided. Among them, different types of adversarial training methods are the most effective. [41] first propose adversarial training as an effective way to improve the robustness of DNNs, and [112] extend it to ensemble adversarial learning. [93] have also proposed robust networks against adversarial examples based on well-defined adversaries. Given the fact that AdvGAN strives to generate adversarial instances from the underlying true data distribution, it can essentially produce more photo-realistic adversarial perturbations compared with other attack strategies. Thus, AdvGAN could have a higher chance to produce adversarial examples that are resilient under different defense methods. In this section, we quantitatively evaluate this property for AdvGAN compared with other attack strategies. Additionally, we argue that stAdv explores the new adversarial space beyond the traditionally $\mathcal{L}_p$-based threat model. Therefore, we also use the adversarial training based defense methods which trained on previous $\mathcal{L}_p$-based adversarial examples to validate correctness of the above argument.

Settings Here we generate adversarial examples in the white-box setting and test different defense methods against these samples to evaluate the strength of these attacks under defenses. We mainly focus on the adversarial training defenses due to their state-of-the-art performance. We apply three defense strategies in our evaluation: the FGSM adversarial training (Adv.) [41], ensemble adversarial training (Ens.) [112], and projectile gradient descent (PGD) adversarial training [93] methods[6]. For each architecture evaluated by our attacks, we apply the above defense methods on them. In detail, we replace f with our the defense models respectively to generate adversarial examples by using AdvGAN and stAdv. We use FGSM and CW as baseline methods for comparison.

The results on the MNIST and CIFAR-10 datasets are shown in Table 2.4. We observe that the three defense strategies can achieve high performance (less than 10% attack success rate) against FGSM and C&W attacks. AdvGAN could achieve a higher attack success rate. It indicates that the GAN structure could benefit the process to find adversarial examples.

However, compare to stAdv, these defense methods only achieve low defense performance on

[6] Each ensemble adversarial trained model is trained using (i) pristine training data, (ii) FGSM adversarial examples generated for the current model under training, and (iii) FGSM adversarial examples generated for naturally trained models of two architectures different from the model under training.

stAdv, which improve the attack success rate to more than 30% among all defense strategies. The potential reason is that the defense methods are trained on $\mathcal{L}_p$-based threat model and stAdv is a new type of adversarial examples which are beyond the scope to $\mathcal{L}_p$-based threat model. On the other side, for stAdv, we do not use $\mathcal{L}_p$ norm to bound the distance as translating an image by one pixel may introduce large $\mathcal{L}_p$ penalty. We instead constrain the spatial transformation flow. Therefore, the previous adversarial trained model for $\mathcal{L}_p$-baesd adversarial examples could not defend against our attack. These results also indicate that a new type of adversarial strategy, such as our spatial transformation-based attack, may open new directions for developing better defense systems.

Table 2.4: Attack success rate of adversarial examples generated by AdvGAN and stAdv under defenses on MNIST and CIFAR-10.

Data	Model	Defense	FGSM	C&W	AdvGAN	stAdv
MNIST	A	Adv.	4.3%	4.6%	**8.0%**	**36.62%**
		Ensemble	1.6%	4.2%	**6.3%**	**48.07%**
		Iter.Adv.	4.4%	2.96%	**5.6%**	**48.38%**
	B	Adv.	6.0%	4.5%	**7.2%**	**50.17%**
		Ensemble	2.7%	3.18%	**5.8%**	**46.14%**
		Iter.Adv.	**9.0%**	3.0%	6.6%	**49.28%**
CIFAR	ResNet	Adv.	13.10%	11.9%	**16.03%**	**43.36%**
		Ensemble.	10.00%	10.3%	**14.32%**	**36.89%**
		Iter.Adv	22.8%	21.4%	**29.47%**	**49.19%**
	Wide ResNet	Adv.	5.04%	7.61%	**14.26%**	**31.66%**
		Ensemble	4.65%	8.43%	**13.94 %**	**29.56%**
		Iter.Adv.	14.9%	13.90%	**20.75%**	**31.60%**

2.5 Conclusion

In this chapter, we propose AdvGAN to generate adversarial examples using generative adversarial networks (GANs). In our AdvGAN framework, once trained, the feed-forward generator can produce adversarial perturbations efficiently. It can also perform both *Semi-whitebox* and black-box attacks with high attack success rate. In addition, when we apply AdvGAN to generate adversarial instances on different models without knowledge of the defenses in place, the generated adversarial examples can attack the state-of-the-art defenses with higher attack success rate than examples generated by the competing methods. This property makes AdvGAN a promising candidate for improving adversarial training defense methods. The generated adversarial examples produced by AdvGAN preserve high perceptual quality due to GANs' distribution approximation property.

Different from the AdvGAN that generate adversarial examples by directly manipulating pixel values, we also introduce a new type of perturbation based on spatial transformation, which aims to

preserve high perceptual quality for adversarial examples. We have shown that adversarial examples generated by stAdv are more difficult for humans to distinguish from original instances. One of the future direction is to propose a general framework to defend against the adversarial examples generated by stAdv. One of the potential way to defend against adversarial examples generated by stAdv is to incorporate them into the training process. To achieve this goal, we still need constrain the manipulation of the flow and give a training budget. Recently, Zhang and Wang [144] introduce a spatial transformation-based attack with budget and further integrate both pixel and spatial attacks into one generation model to improving the overall model robustness against both attacks. However, how to defend against spatial transformed adversarial examples without budget is still an open problem now.

CHAPTER 3

Adversarial Examples in the 3D Space

3.1 Introduction

The previous chapter introduced adversarial examples in the 2D digital space. Such digital space studies still leaves open the question of the existence of adversarial examples in the physical space. Digital attacks generated by directly manipulating pixels can be defended by securing the camera, so that the generated images are not realizable in practice. Understanding adversarial examples in the physical/3D world is an important step towards the resilient learning algorithms and there has been significant prior progress on generating physically possible adversarial examples [5, 13, 38, 72] by altering the texture of a 3D surface, *i.e*. applying adversarial printable 2D patches or painting patterns.Such attacks, however, are less suitable for textureless objects, because adding texture to an otherwise textureless surface may increase the chance of being detected and defended. Additionally, for the texture-ignore cyber physical equipment such as LiDAR and RaDAR, the texture-based perturbations will not be collected resulting in the ineffectiveness and ambiguity of the existence problem of adversarial perturbation in the physical world. Digital images are formed by the processing of illumination, shape and texture. Therefore, in this chapter, we explore a new avenue of attacks where we generate physically possible adversarial examples by simulating the photo-taken process. We explore 3D objects that have rich shape features but minimal texture variation, and show that we can still fulfill the adversarial goal by perturbing the shape of those 3D objects, while the same method can still be applied to textures. Specifically, we propose meshAdv to generate adversarial meshes with negligible perturbations. We leverage a physically-based differentiable renderer [66] to accurately render the mesh under the certain camera and lighting parameters. A deep network then outputs a prediction given the rendered image as input. Since this whole process is differentiable, gradients can be propagated from the network prediction back to the shape or texture of the mesh. Therefore, we can use gradient-based optimization to generate shape-based or texture-based perturbation by applying losses on the network output. The entire pipeline is shown in Figure 3.1.

Even though we are only manipulating physical properties (shape and texture) of a 3D object,

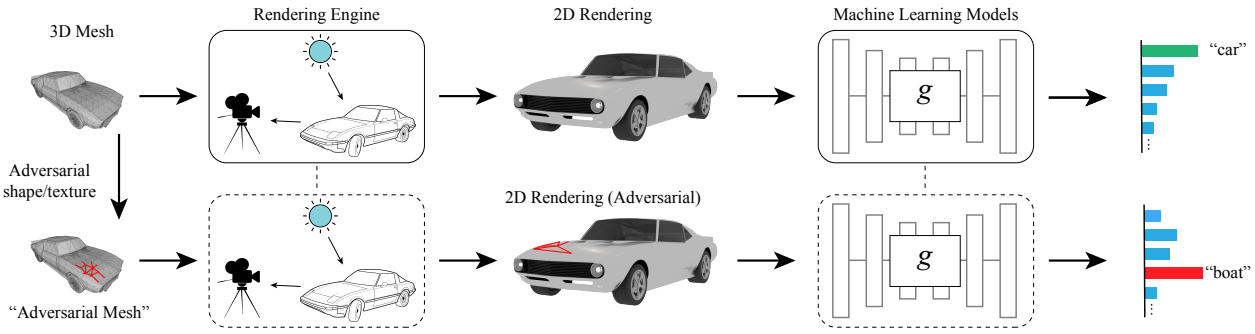


Figure 3.1: The pipeline of "adversarial mesh" generation by meshAdv.

we can always fool state of the art DNNs (see Section 3.5.2). Specifically, we show that for any fixed rendering conditions (*i.e.* lighting and camera parameters), state of the art object classifiers (DenseNet [54] and Inception-v3 [109]) and detector (Yolo-v3 [98]) can be consistently tricked by slightly perturbing the shape and texture of 3D objects. We further show that by using multiple views optimization, the attack success rate of "adversarial meshes" increases under various viewpoints (see Table 3.2). In addition, we conduct user studies to show that the generated perturbations are negligible to human perception.

Since the perturbation on meshes is adversarially optimized with the help of a differentiable renderer, a natural question to ask is whether a similar method can be applied in practice when the rendering operation is not differentiable and even the renderer parameters are known. We try to answer this question by proposing a pipeline to perform black-box attack on a photo-realistic renderer (with non-differentiable rendering operation) under unknown rendering parameters. We show that via estimating the rendering parameters and improving the robustness of perturbation, our generated "adversarial meshes" can attack a photorealistic renderer.

Additionally, we visualize our shape based perturbation to show possible vulnerable regions for meshes. This can be beneficial when we hope to improve the robustness (against shape deformation) of machine learning models that are trained on 3D meshes [20, 104, 124] for different tasks such as viewpoint estimation [106], indoor scene understanding [46, 89, 104, 147] and so on [23, 88, 99, 116, 136].

To summarize, our contributions in this chapter are as follows:

- We propose an end-to-end optimization-based method stAdv to generate 3D "adversarial meshes" with negligible perturbations, and show that it is effective in attacking different machine learning tasks;
- We demonstrate the effectiveness of our method on a black-box non-differentiable renderer

with unknown parameters;

- We provide insights into vulnerable regions of a mesh via visualizing the flow of shape based perturbations;
- We conduct user studies to show that our 3D perturbation is subtle enough and will not affect user recognition.

We organize this chapter as follows. We introduce the problem definition in Section 3.2, challenges in Section 3.2, and methodology in Section 3.3. We show how to attack a non-differentiable renderer in section 3.4.

3.2 Problem Definition and Challenges

Similar to the formation in the previous chapter, in the 2D domain, let f be a machine learning model trained to map a 2D image x to its category label y. For f, an adversarial attacker targets to generate an adversarial image x_{adv} such that $f(x_{adv}) \neq y$ (untargeted attack) or $g(x_{adv}) = t$ (targeted attack), where y is the ground truth label and t is our specified malicious target label.

Unlike adversarial attacks in 2D space, here the image x is a rendered result of a 3D object S: $x = R(S; P, L)$, computed by a physically-based renderer R with camera parameters P and illumination parameters L. In other words, it is not allowed to directly operate the pixel values of I, and one has to manipulate the 3D object S to generate S_{adv} such that the rendered image of it will fool f to make incorrect predictions: $x_{adv} = R(S_{adv}; P, L)$.

Achieving the above goals is non-trivial due to the following challenges: 1) **Manipulation space**: When rendering 3D contents, shape, texture and illumination are entangled together to generate the pixel values in a 2D image, so image pixels are no longer independent with each other. This means the manipulation space can be largely reduced due to image parameterization. 2) **Constraints in 3D**: 3D constraints such as physically possible shape geometry and texture are not directly reflected on 2D [142]. Human perception of an object is in 3D or 2.5D [87], and perturbation of shape or texture on 3D objects may directly affect human perception of them. This means it can be challenging to generate unnoticeable perturbations on 3D meshes.

3.3 Methodology

Here we assume the renderer R is known (*i.e.* white box) and differentiable to the input 3D object S in mesh representation. To make a renderer R differentiable, we have to make several assumptions regarding object material, lighting models, interreflection $etc.$ With a differentiable renderer, we can use gradient-based optimization algorithms to generate the mesh perturbations in an end-to-end manner, and we denote this method meshAdv.

3.3.1 Differentiable Rendering

A physically-based renderer R computes a 2D image $x = R(S; P, L)$ with camera parameters P, a 3D object S and lighting parameters L by approximating physics, *e.g.* the rendering equation [57, 64]. A differentiable renderer makes such computation differentiable w.r.t. the input S, P, L by making assumptions on illumination models and surface reflectance, and simplifying the ray-casting process. Following common practice, we use 3D triangular meshes for object representation, Lambertian surface for surface modeling, directional lighting with a uniform ambient for illumination, and ignore interreflection and shadows. Here, we further explain the details regarding 3D mesh representation $S = (V, F, T)$, illumination model L and camera parameters P used in differentiable rendering in this work.

For a 3D object S in 3D triangular mesh representation, let V be the set of its n vertices in 3D space, and F be the indices of its m faces:

$$V = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n \in \mathbb{R}^3\}, \qquad F = \{\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_m \in \mathbb{N}^3\} \tag{3.1}$$

For textures, traditionally, they are represented by 2D texture images and mesh surface parameterization such that the texture images can be mapped onto the mesh's triangles. For simplicity, here we attach to each triangular face a single RGB color as its reflectance:

$$T = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots, \boldsymbol{t}_m \in \mathbb{R}^{+3}\} \tag{3.2}$$

For the illumination model, we use k directional light sources plus an ambient light. The lighting directions are denoted L_{dir}, and the lighting colors (in RGB color space) are denoted as L_{color} for directional light sources and $\boldsymbol{a}$ for the ambient light:

$$L_{\text{dir}} = \{\boldsymbol{l}_1^d, \boldsymbol{l}_2^d, \cdots \boldsymbol{l}_k^d \in \mathbb{R}^3\}, \qquad L_{\text{color}} = \{\boldsymbol{l}_1^c, \boldsymbol{l}_2^c, \cdots \boldsymbol{l}_k^c \in \mathbb{R}^3\} \tag{3.3}$$

We put the mesh $S = (V, F, T)$ at the origin $(0, 0, 0)$, and set up our perspective camera following a common practice: the camera viewpoint is described by a quadruple $P = (d, \theta, \phi, \psi)$, where d is the distance of the camera to the origin, and θ, ϕ, ψ are azimuth, elevation and tilt angles respectively. Note that here we assume the camera intrinsics are fixed and we only need gradients for the extrinsic parameters P.

Given the above description, the 2D image produced by the differentiable renderer can be symbolized as follows:

$$x = \text{rasterize}(P, T \cdot \text{shading}(L, \text{normal}(V, F))) \tag{3.4}$$

normal(·, ·) computes the normal direction $\boldsymbol{n}_i$ for each triangular face $\boldsymbol{f}_i$ in the mesh, by computing the cross product of the vectors along two edges of the face:

$$\boldsymbol{n}_i = \frac{(\boldsymbol{v}_{\boldsymbol{f}_i^{(1)}} - \boldsymbol{v}_{\boldsymbol{f}_i^{(2)}}) \times (\boldsymbol{v}_{\boldsymbol{f}_i^{(2)}} - \boldsymbol{v}_{\boldsymbol{f}_i^{(3)}})}{\left\| (\boldsymbol{v}_{\boldsymbol{f}_i^{(1)}} - \boldsymbol{v}_{\boldsymbol{f}_i^{(2)}}) \times (\boldsymbol{v}_{\boldsymbol{f}_i^{(2)}} - \boldsymbol{v}_{\boldsymbol{f}_i^{(3)}}) \right\|_2} \tag{3.5}$$

shading computes the shading intensity $vs._i$ on the face given the face normal direction $\boldsymbol{n}_i$ and lighting parameters:

$$vs._i = \boldsymbol{a} + \sum_{i=1}^{k} \boldsymbol{l}_i^c \max(\boldsymbol{l}_i^d \cdot \boldsymbol{n}_i, 0) \tag{3.6}$$

Given face reflectance $\boldsymbol{t}_i$ for each face i, we compute the color $\boldsymbol{c}_i$ of each face i by elementwise multiplication:

$$\boldsymbol{c}_i = \boldsymbol{t}_i \circ vs._i \tag{3.7}$$

rasterize projects the computed face colors $\boldsymbol{c}_i$ in 3D space onto the 2D camera plane by raycasting and depth testing. We also cap the color values to $[0, 1]$.

For implementation, we use the off-the-shelf PyTorch implementation [68, 97] of the Neural Mesh Renderer (NMR) [66].

3.3.2 Optimization Objective

Here, instead of optimizing the input image x, we optimize the S.

$$\mathcal{L}(S_{adv}) = \mathcal{L}_{\text{adv}}(S_{adv}, f) + \lambda \mathcal{L}_{\text{perceptual}}(S_{adv}) \tag{3.8}$$

In this equation, $\mathcal{L}_{\text{adv}}$ is the adversarial loss to fool the model f, given the rendered image $x^{\text{adv}} = R(S^{\text{adv}}; P, L)$ as input. $\mathcal{L}_{\text{perceptual}}$ is the loss to keep the "adversarial mesh" perceptually realistic. λ is a balancing hyper-parameter.

We further instantiate $\mathcal{L}_{\text{adv}}$ and $\mathcal{L}_{\text{perceptual}}$ in the next subsections, regarding different tasks (classification or object detection) and perturbation types (shape or texture).

3.3.2.1 Adversarial Loss

Classification For a classification model f, the output is usually the probability distribution of object categories, given an image of the object as the input. We use the cross entropy loss [31] as the adversarial loss for meshAdv:

$$\mathcal{L}_{\text{adv}}(S_{\text{adv}}; g, t) = t \log(g(x_{adv})) + (1 - t) \log(1 - g(x_{adv})), \tag{3.9}$$

where $x_{adv} = R(S_{\text{adv}}; P, L)$, and t is one-hot representation of the target label.

Object Detection For object detection, we choose a state-of-the-art model Yolo-v3 [98] as our victim model. It divides the input image x into $Z \times Z$ different grid cells. For each grid cell, Yolo-v3 predicts the locations and label confidence values of B bounding boxes. For each bounding box, it generates 5 values (4 for the coordinates and 1 for the objectness score) and a probability distribution over N classes. Here the adversary's goal is to make the victim object disappear from the object detector, called *disappearance attack*. So we use the disappearance attack loss [39] as our adversarial loss for Yolo-v3:

$$\mathcal{L}_{\text{adv}}(S_{adv}; g, t) = \max_{z \in Z^2, b \in B} H(z, b, t, g(x_{adv})), \tag{3.10}$$

where $x_{adv} = R(S_{adv}; P, L)$, and $H(\cdot)$ is a function to represent the probabilities of label t for bounding box b in the grid cell z, given I^{adv} as the input of the model g.

3.3.2.2 Perceptual Loss

To keep the "adversarial mesh" perceptually realistic, we leverage a Laplacian loss similar to the total variation loss [118] as our perceptual loss:

$$\mathcal{L}_{\text{perceptual}}(S_{adv}) = \sum_i \sum_{q \in \mathcal{N}(i)} \left\| x^i_{adv} - x^q_{adv} \right\|_2^2, \tag{3.11}$$

where x^i is the RGB vector of the i-th pixel in the image $x_{adv} = R(S_{adv}; P, L)$, and $\mathcal{N}(i)$ is the 4-connected neighbors of pixel i.

We apply this smoothing loss to the image when generating texture based perturbation for S_{adv}. However, for shape based perturbation, manipulation of vertices may introduce unwanted mesh topology change, as reported in [66]. Therefore, instead of using Eq. (3.11), we perform smoothing on the displacement of vertices such that neighboring vertices will have a similar displacement flow. We achieve this by extending the smoothing loss to 3D vertex flow, in the form of a Laplacian loss:

$$\mathcal{L}_{\text{perceptual}}(S_{adv}) = \sum_{\boldsymbol{v}^i \in V} \sum_{\boldsymbol{v}^q \in \mathcal{N}(\boldsymbol{v}^i)} \left\| \Delta \boldsymbol{v}^i - \Delta \boldsymbol{v}^q \right\|_2^2, \tag{3.12}$$

where $\Delta \boldsymbol{v}_i = \boldsymbol{v}^i_{\text{adv}} - \boldsymbol{v}^i$ is the displacement of the perturbed vertex $\boldsymbol{v}^i_{\text{adv}}$ from its original position $\boldsymbol{v}^i$ in the pristine mesh, and $\mathcal{N}(\boldsymbol{v}^i)$ denotes connected neighboring vertices of $\boldsymbol{v}^i$ defined by mesh triangles.

3.4 Transferability to Black-Box Renderers

Our meshAdv aims to white-box-attack the system $f(R(S; P, L))$ by optimizing S end-to-end since R is differentiable. However, we hope to examine the potential of meshAdv for 3D objects in practice, where the actual renderer may be unavailable.

We formulate this as a black-box attack against a non-differentiable renderer R' under unknown rendering parameters P^*, L^*, e.g. we have limited access to R' but we still want to generate S_{adv} such that $R'(S_{adv}, P^*, L^*)$ fools the model f. Because we have no assumptions on the black-box renderer R', it can render photorealistic images at a high computational cost, by enabling interreflection, occlusion and rich illumination models etc such that the final image is an accurate estimate under real-world physics as if captured by a real camera. In this case, the transferability of "adversarial meshes" generated by meshAdv is crucial since we want to avoid the expensive computation in R' and still be able to generate such S_{adv}. We analyze two scenarios for such transferability.

Controlled Rendering Parameters Before black-box attacks, we want to first test our "adversarial meshes" directly under the same rendering configuration (lighting parameters L, camera parameters P), only replacing the differentiable renderer R with the photorealistic renderer R'. In other words, while $x_{adv} = R(S_{adv}; P, L)$ can fool the model f as expected, we would like to see whether $x_{adv} = R'(S_{adv}; P, L)$ can still fool the model f.

Unknown Rendering Parameters In this scenario, we would like to use meshAdv to attack a non-differentiable system $g(R'(S; P^*, L^*))$ under fixed, unknown rendering parameters P^*, L^*. In practice, we will have access to the mesh S and its mask M in the original photorealistic rendering $x' = R'(S; P^*, L^*)$, as well as the model f. Directly transfer from one renderer to another may not work due to complex rendering conditions. To improve the performance of such black-box attack, we propose a pipeline as follows:

1. Estimate camera parameters $\hat{P}$ by reducing the error of object silhouette $\|R_{\text{mask}}(S; P) - M\|^2$, where $R_{\text{mask}}(S; P)$ renders the mask of the object S (lighting is irrelevant to produce the mask);

2. Given $\hat{P}$, estimate lighting parameters $\hat{L}$ by reducing the masked error of rendered images: $\left\|M \circ (R(S; \hat{P}, L) - I')\right\|^2$, where the operator $\circ$ is Hadamard product;

3. Given $\hat{P}, \hat{L}$, use meshAdv to generate the "adversarial mesh" S_{adv} such that $R(S_{adv}; \hat{P}, \hat{L})$ fools g; To improve robustness, we add random perturbations to $\hat{P}$ and $\hat{L}$ when optimizing;

4. Test S_{adv} in the original scene with photorealistic renderer R': obtain the prediction $g(R'(S_{adv}; P^*, L^*))$.

3.5 Experimental Results

In this section, we first show the attack effectiveness of "adversarial meshes" generated by meshAdv against classifiers under different settings. We then visualize the perturbation flow of

vertices to better understand the vulnerable regions of those 3D objects. User studies show that the proposed perturbation is subtle and will not mislead human recognition. In addition, we show examples of applying meshAdv to object detectors in physically realistic scenes. Finally, we evaluate the transferability of "adversarial meshes" generated by meshAdv and illustrate how to use such transferability to attack a black-box renderer.

3.5.1 Experimental Setup

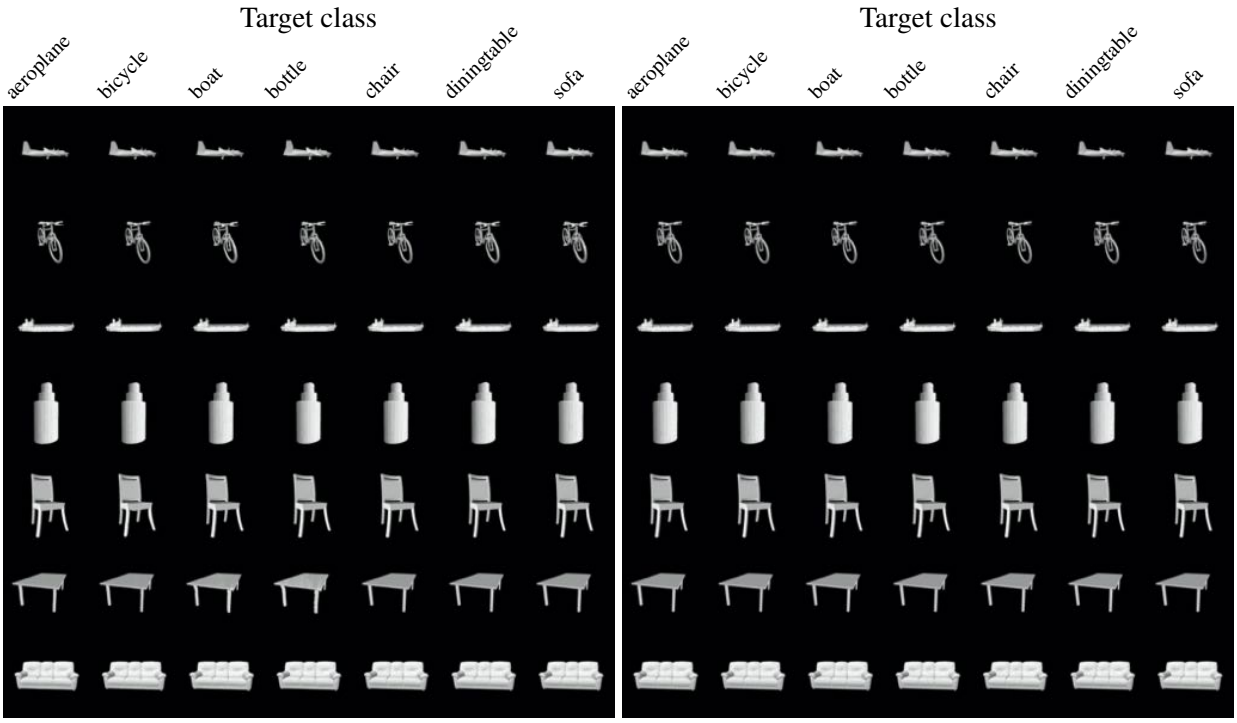
Perturbation Type	Model	Test Accuracy	Best Case		Average Case		Worst Case	
			Avg. Distance	Succ. Rate	Avg. Distance	Succ. Rate	Avg. Distance	Succ. Rate
Shape	DenseNet	100.0%	8.4×10^{-5}	100.0%	1.8×10^{-4}	100.0%	3.0×10^{-4}	100.0%
	Inception-v3	100.0%	4.8×10^{-5}	100.0%	1.2×10^{-4}	99.8%	2.3×10^{-4}	98.6%
Texture	DenseNet	100.0%	3.8×10^{-3}	100.0%	1.1×10^{-2}	99.8%	2.6×10^{-2}	98.6%
	Inception-v3	100.0%	3.7×10^{-3}	100.0%	1.3×10^{-2}	100.0%	3.2×10^{-2}	100.0%

Table 3.1: Attack success rate of meshAdv and average distance of generated perturbation for different models and different perturbation types. We choose rendering configurations in *PASCAL3D+ renderings* such that the models have 100% test accuracy on pristine meshes so as to confirm the adversarial effects. The average distance for shape based perturbation is computed using the 3D Laplacian loss from Equation 3.12. The average distance for texture based perturbation is the root-mean-squared error of face color change.

For victim learning models f, we choose DenseNet [54] and Inception-v3 [109] trained on ImageNet [32] for classification, and Yolo-v3 trained on COCO [77] for object detection. For meshes (S), we preprocess CAD models in PASCAL3D+ [126] using uniform mesh resampling with MeshLab [25] to increase triangle density. Those meshes are then scaled to $[-1, 1]$ and put into the scene. Then, we use Neural Mesh Renderer (NMR) to generate synthetic renderings using these unitized meshes with uniformly sampled random camera parameters: azimuth from $[0°, 360°)$, elevation from $[0°, 90°]$. As for lighting, we used a directional light and an ambient light for *PASCAL3D+ Renderings*. The direction is uniformly sampled in a cone such that the angle between the view and the lighting direction is less than $60°$.

In order to obtain the groundtruth labels, we map the object classes in PASCAL3D+ to the corresponding classes in the ImageNet. Next, we feed the synthetic renderings to DenseNet and Inception-v3 and filter out the samples that are misclassified by either network, so that both models have 100% prediction accuracy on our *PASCAL3D+ renderings*. We then save the rendering configurations for evaluation of meshAdv.

For the differentiable renderer (R), we use the off-the-shelf PyTorch implementation [68, 97] of the Neural Mesh Renderer (NMR) [66] to generate "adversarial meshes". For rendering settings ($R(\cdot; P, L)$) when attacking classifiers, we randomly sample camera parameters P and lighting



(a) Perturbation on shape (b) Perturbation on texture

Figure 3.2: Benign images (diagonal) and corresponding adversarial examples generated by meshAdv on *PASCAL3D+ renderings* tested on Inception-v3. Adversarial target classes are shown at the top. We show perturbation on (a) shape and (b) texture.

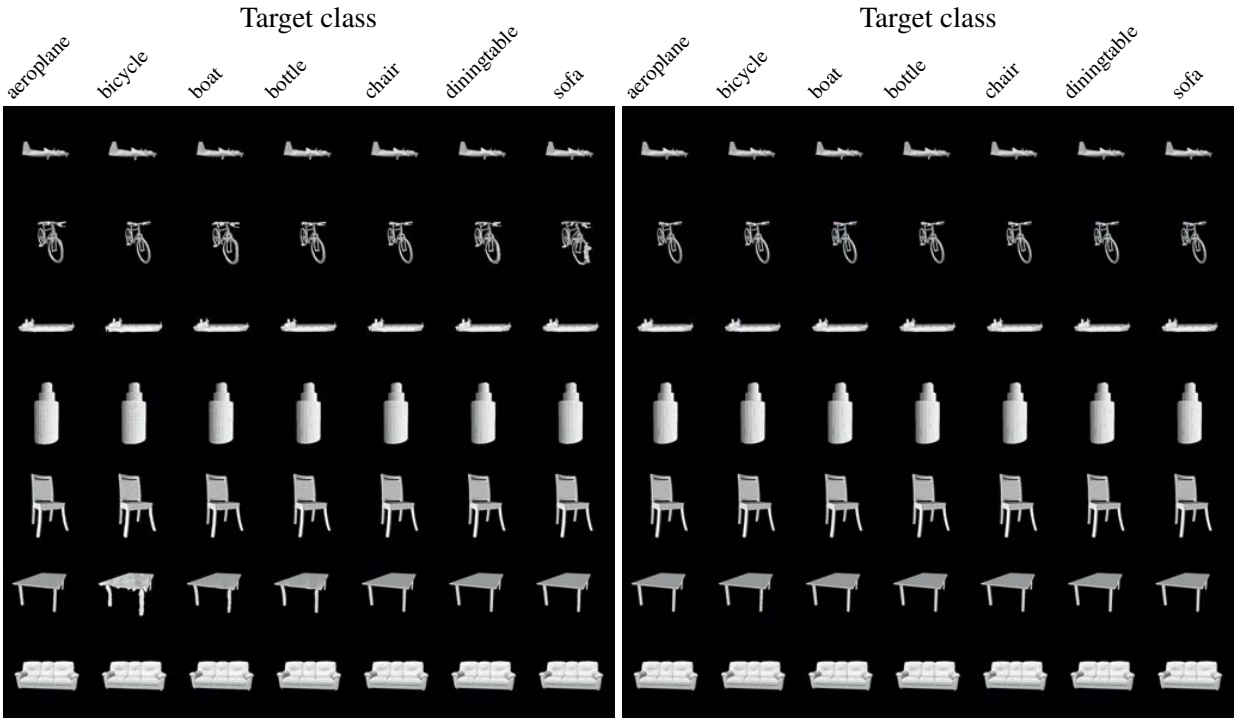
parameters L, and filter out configurations such that the classification models have 100% accuracy when rendering pristine meshes. These rendering configurations are then fixed for evaluation, and we call meshes rendered under these configurations *PASCAL3D+ renderings*. In total, we have 7 classes, and for each class we generate 72 different rendering configurations.

For optimizing the objective, we use Adam [67] as our solver. In addition, we select the hyperparameter λ in Equation 3.8 using binary search, with 5 rounds of search and 1000 iterations for each round.

3.5.2 meshAdv on Classification

In this section, we evaluate the quantitative and qualitative performance of meshAdv against classifiers.

For each sample in our *PASCAL3D+ renderings*, we try to targeted-attack it into the other 6 categories. Next, for each perturbation type (shape and texture) and each model (DenseNet and Inception-v3), we split the results into three different cases similar to [18]: *Best Case* means we attack samples within one class to other classes and report on the target class that is *easiest* to attack. *Average Case* means we do the same but report the performance on *all* of the target



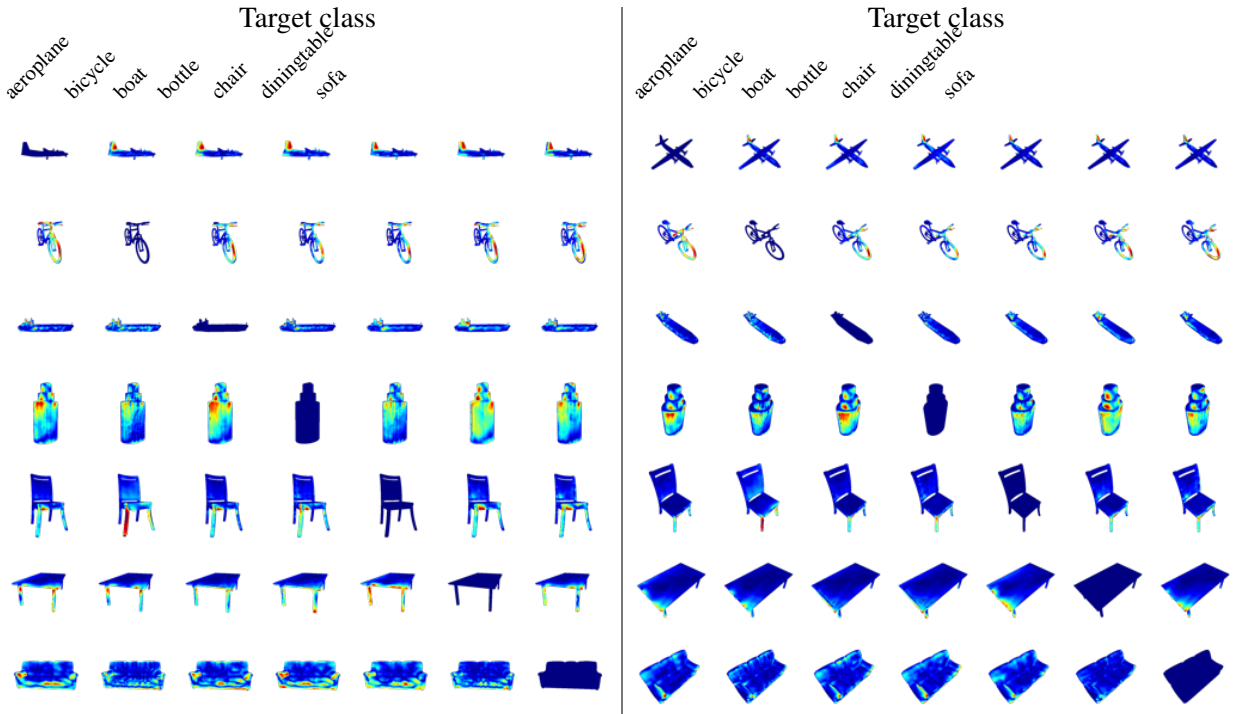
(a) Perturbation on shape

(b) Perturbation on texture

Figure 3.3: Benign images (diagonal) and corresponding adversarial examples generated by meshAdv on *PASCAL3D+ renderings* tested on Inception-v3. Adversarial target classes are shown at the top. We show perturbation on (a) shape and (b) texture.

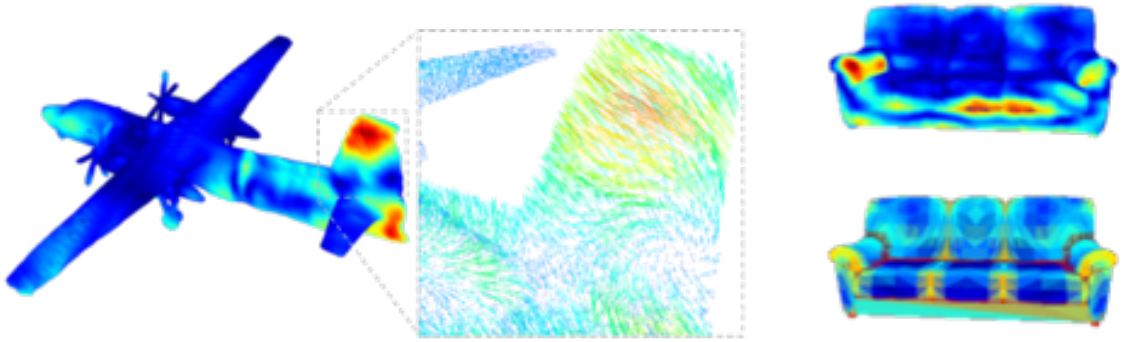
classes. Similarly, *Worst case* means that we report on the target class that is *hardest* to attack. The corresponding results are shown in Table 3.1, including attack success rate of meshAdv, and the evaluation on generated shape and texture based perturbation respectively. For shape based perturbation, we use the Laplacian loss from Equation 3.12 as the distance metric. For texture based perturbation, we compute the root-mean-square distance of texture values for each face of the mesh: $\sqrt{\frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{t}_i^{\text{adv}} - \boldsymbol{t}_i)^2}$, where $\boldsymbol{t}_i$ is the texture color of the i-th face among the mesh's total m faces. The results show that meshAdv can achieve an almost 100% attack success rate for either adversarial perturbation types.

Figure 3.2 shows the generated "adversarial meshes" against Inception-v3 after manipulating the vertices and texture respectively. The diagonal shows the images rendered with the pristine meshes. The target class of each "adversarial mesh" is shown at the top, and similar results for DenseNet are included in Figure 3.3. Note that the samples in the image are randomly selected and not manually curated. It is worth noting that the perturbation on object shape or texture, generated by meshAdv, is barely noticeable to human, while being able to mislead classifiers. To help assess the vertex displacement in shape perturbation, we discuss the flow visualization and human perceptual study in the following paragraphs.



(a) Rendered view

(b) Canonical view



(c) Vertex flow of an "adversarial mesh" targeting "bicycle"

(d) Perturbation (top) vs curvature (bottom)

Figure 3.4: (a) and (b) are visualization of shape based perturbation with respect to Figure 3.2(a). (c) is a close view of flow directions, and (d) is an example to compare the magnitude of perturbation with the magnitude of curvature. Warmer color indicates greater magnitude and vice versa.

Visualizing Vertex Manipulation In order to better understand the vulnerable regions of 3D objects, in Figure 3.4, we visualize the magnitude of the vertex manipulation flow using heatmaps. The heatmaps in the figure correspond to the ones in Figure 3.2(a). We adopt two viewpoints in this figure: the rendered view (i), which is the same as the one used for rendering the images; and the canonical view (ii), which is achieved by fixing camera parameters for all shapes: we set the azimuth angle to 135° and the elevation angle to 45°. From the heatmaps we observe that the regions with large curvature value and close to the camera (such as edges) are more vulnerable, as

Victim Set Size	Azimuth Range		
	$45^\circ \sim 60^\circ$	$35^\circ \sim 70^\circ$	$15^\circ \sim 75^\circ$
5 views	67%	45%	28%
10 views	73%	58%	38%
15 views	79%	74%	48%

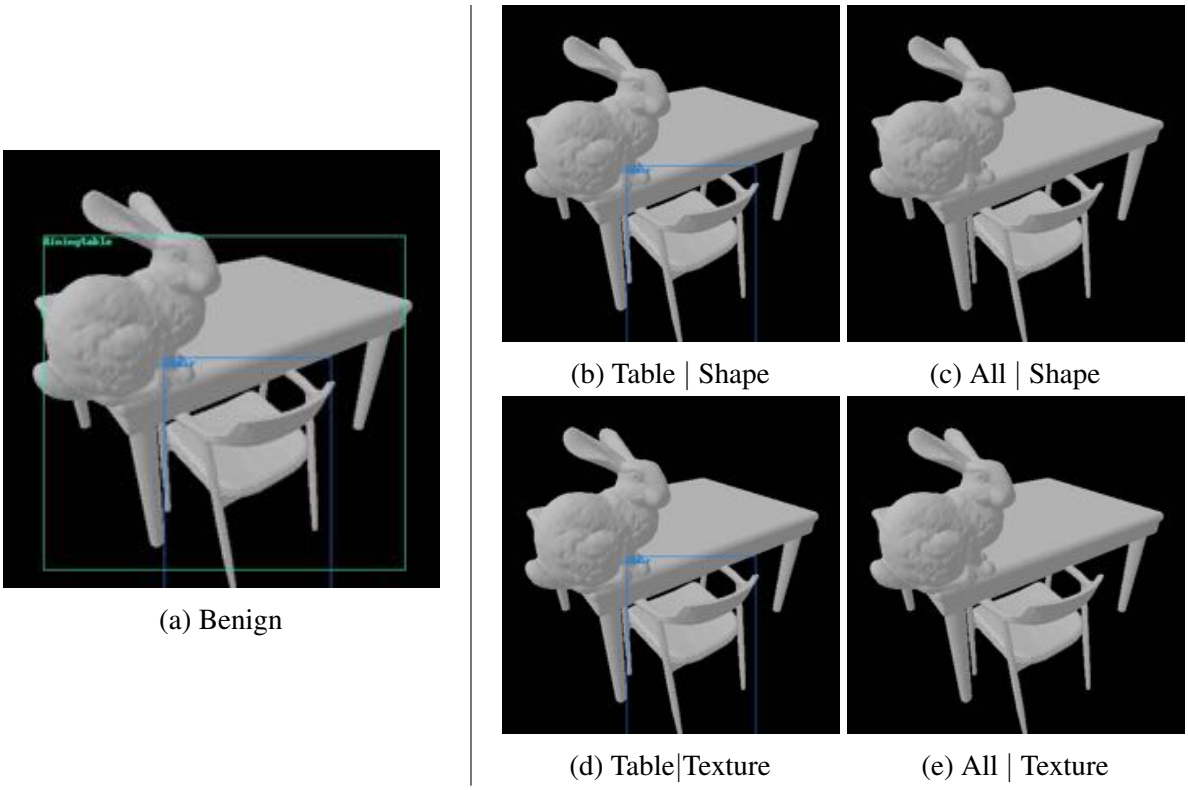
Table 3.2: Targeted attack success rate for unseen camera views. We attack using 5, 10, or 15 views, and test with 20 unseen views in the same range.

shown in the example in Figure 3.4(d). We find this is reasonable, since vertex displacement in those regions will bring significant change to normals, thus affecting the shading from the light sources and causing the screen pixel value to change drastically.

In addition to magnitude, we additionally show an example of flow directions in Figure 3.4(c), which is a close-up 3D quiver plot of the vertex flow in the vertical stabilizer region of an aeroplane. In this example, the perturbed aeroplane mesh is classified as "bicycle" in its rendering. From this figure, we observe that the adjacent vertices tend to flow towards similar directions, illustrating the effect of our 3D Laplacian loss operated on vertex flows (Equation 3.12).

Human Perceptual Study We conduct a user study on Amazon Mechanical Turk (AMT) in order to quantify the realism of the adversarial meshes generated by meshAdv. We uploaded the adversarial images which are misclassified by DenseNet and Inception-v3. Participants were asked to recognize those adversarial object to one of the two classes (the ground-truth class and the adversarial target class). The order of these two classes was randomized and the adversarial objects are appeared for 2 seconds in the middle of the screen during each trial. After disappearing, the participant has unlimited time to select the more feasible class according to their perception. For each participant, one could only conduct at most 50 trials, and each adversarial image was shown to 5 different participants. In total, we collect 3820 annotations from 49 participants. In $99.29 \pm 1.96\%$ of trials the "adversarial meshes" were recognized correctly, indicating that our adversarial perturbation will not mislead human as they can almost always assign the correct label of these "3D adversarial meshes".

Multiview Robustness Analysis In addition to a fixed camera when applying meshAdv, we also explore the robustness of meshAdv against a range of viewpoints for shape based perturbation. First, we create a victim set of images rendered under 5, 10 or 15 different azimuth angles for optimizing the attack. We then sample another 20 unseen views within the range for test. The results are shown in Table 3.2. We can see that the larger the azimuth range is, the harder to achieve a high attack success rate. In the meantime, meshAdv can achieve a relatively high attack success rate when



(a) Benign

(b) Table | Shape

(c) All | Shape

(d) Table|Texture

(e) All | Texture

Figure 3.5: "Adversarial meshes" generated by meshAdv in a synthetic indoor scene. (a) represents the benign rendered image and (b)-(e) represent the rendered images from "adversarial meshes" by manipulating the shape or texture. We use the format "adversarial target | perturbation type" to denote the victim object aiming to hide and the type of perturbation respectively.

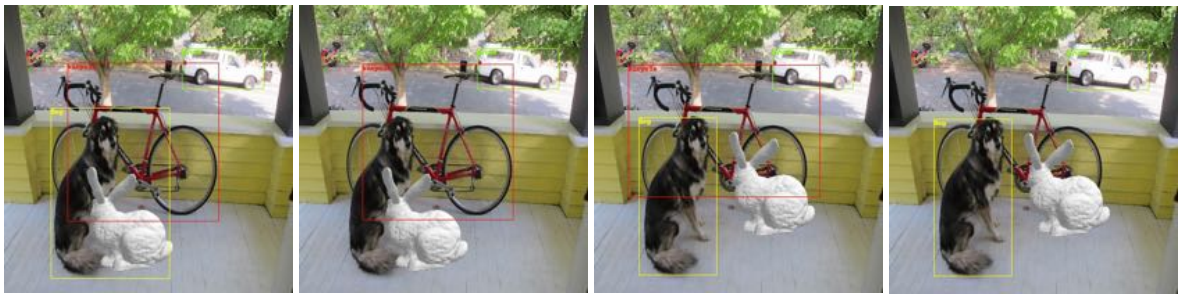
more victim instances are applied for training. As a result, it shows that the attack robustness can potentially be improved under various viewpoints by optimizing on a large victim set.

3.5.3 meshAdv on Object Detection

For object detection, we use Yolo-v3 [98] as our target model.

Indoor Scene First, we test meshAdv within the indoor scene which is pure synthetic. We compose the scene manually with a desk and a chair to simulate an indoor setting, and place in the scene a single directional light with low ambient light. We then put the Stanford Bunny mesh [115] onto the desk, and show that by manipulating either the shape or the texture of the mesh, we can achieve the goal of either removing the target table detection or removing all detections while keeping the perturbation almost unnoticeable, as shown in Figure 3.5.

Outdoor Scene Given a real photo of an outdoor scene, we hope to remove the detections of real objects in the photo. Different from the indoor sceen in which lighting is known, we have to



(a) S | GT (b) S_{adv} | Dog (c) S | GT (d) S_{adv}|Bicycle

Figure 3.6: "Adversarial meshes" generated by meshAdv for an outdoor photo. (a) and (c) show images rendered with pristine meshes as control experiments, while (b) and (d) contain "adversarial meshes" by manipulating the shape. We use the format " S/S_{adv} | target" to denote the benign/adversarial 3D meshes and the target to hide from the detector respectively.

estimate the parameters of a sky lighting model [52] using the API provided by [51] as groundtruth lighting and adapt to the differentiable renderer. We then use this lighting to render our mesh onto the photo. In the real photo, we select the dog and the bicycle as our target objects and aim to remove the detection one at a time. We show that we successfully achieve the adversarial goal with barely noticeable perturbation, as in Figure 3.6.

3.5.4 Transferability to Black-Box Renderers

As mentioned in Section 3.4, the final adversarial goal is to black-box attack a system $g(R'(S; P, L))$ in which the renderer R' is a computationally intensive renderer that is able to produce photorealistic images. Here we choose Mitsuba [60] as such renderer, and focus on shape based perturbation.

Controlled Rendering Parameters Before perform such attacks, we first evaluate the transferability under controlled parameters. We directly render the "adversarial meshes" S^{adv} generated in Section 3.5.2 using Mitsuba, with the same lighting and camera parameters. We then calculate the targeted/untargeted attack success rate by feeding the Mitsuba-rendered images to the same victim classification models g. The result of untargeted attacks are shown in Table 3.3, and the confusion matrices for targeted attacks are show in Figure 3.7. We observe that for untargeted attack, the "adversarial meshes" can be transferred to Mitsuba with relatively high atttack success rate for untargeted attack; while as shown in Figure 3.7, the targeted attack barely transfers in this straightforward setting.

Unknown Rendering Parameters To more effectively targeted attack the system $f(R'(S; P^*, L^*))$ when rendering parameters P^*, L^* are unknown, we apply the pipeline from Section 3.4 on a classifier and an object detector, respectively. we first use the Adam optimizer [67] to obtain the camera estimate $\hat{P}$, then estimate the lighting L^* using 5 directional lights and an ambient light $\hat{L}$. Note that

Model/Target	aeroplane	bicycle	boat	bottle
DenseNet	65.2%	69.1%	66.7%	63.0%
Inception-v3	67.1%	83.3%	39.6%	76.9%
Model/Target	chair	diningtable	sofa	**average**
DenseNet	37.1%	70.3%	47.9%	59.8%
Inception-v3	32.1%	75.0%	52.3%	60.9%

Table 3.3: Untargeted attack success rate against Mitsuba by transferring "adversarial meshes" generated by attacking a differentiable renderer targeting different classes.

Figure 3.7: Confusion matrices of targeted success rate for evaluating transferability of "adversarial meshes" on different classifiers. **Left**: DenseNet; **right**: Inception-v3.

the groundtruth lighting L^* spatially varies due to interreflection and occlusion, so it is impossible to have an exact estimate using the global lighting model in NMR. Then we manipulate the shape S_{adv} in the NMR until the image $x_{adv} = R(S_{\text{adv}} : \hat{P}, \hat{L})$ can successfully targeted-attack the classifier or the object detector g with a high confidence. During this process, we add small random perturbation to the estimated parameters $(\hat{P}, \hat{L})$ such that S^{adv} will be more robust under uncertainties. For testing, we re-render S_{adv} with Mitsuba using the original setting and test the rendered image $x'_{\text{adv}} = R'(S_{\text{adv}}, P^*, L^*)$ on the same model f.

For classification, we place an aeroplane object from PASCAL3D+ and put it in an outdoor scene under sky light. As is shown in Figure 3.8, we successfully attacked the classifier to output the target "hammerhead" by replacing the pristine mesh with our "adversarial mesh" in the original scene. Note that even we do not have an accurate lighting estimate, we still achieve the transferability by

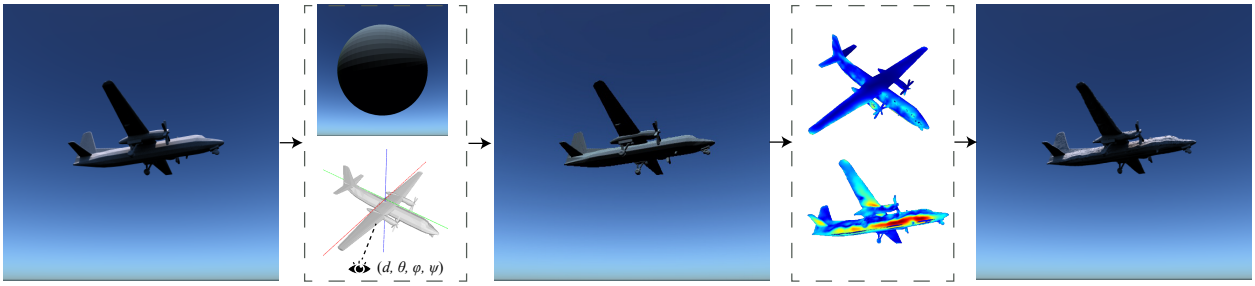
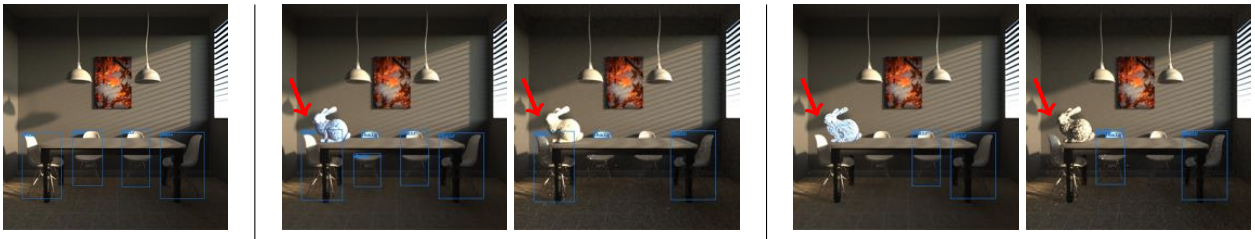


Figure 3.8: Transferability of "adversarial meshes" against classifiers in unknown rendering environment. We estimate the camera viewpoint and lighting parameters using the differentiable renderer NMR, and apply the generated "adversarial mesh" to the photorealistic renderer Mitsuba. The "airliner" is misclassified to the target class "hammerhead" after rendered by Mitsuba.



(a) Benign | (b) S | NMR (c) S | Mit. | (d) S_{adv} | NMR (e) S_{adv} | Mit.

Figure 3.9: Transferability of "adversarial meshes" against object detectors in unknown rendering environment. (b) (c) are controlled experiments. S^{adv} is generated using NMR (d), targeting to hide the leftmost chair (see red arrows), and the adversarial mesh is tested on Mit. (Mitsuba) (e). We use "S/S^{adv} | renderer" to denote whether the added object is adversarially optimized and the renderer that we aim to attack with transferability respectively.

adding perturbation to lighting parameters. For object detection, we modified a scene from [10], and placed the Stanford Bunny object into the scene. The adversarial goal here is to remove the *leftmost* chair in the image. Without an accurate lighting estimate, Figure 3.9 shows that the "adversarial meshes" can still successfully remove the target (the leftmost chair) from the detector.

3.6 Conclusion

In this chapter, we proposed meshAdv to generate "adversarial meshes" by manipulating the shape or the texture of a mesh. These "adversarial meshes" can be rendered to 2D domains to mislead different machine learning models. We evaluate meshAdv quantitatively and qualitatively using CAD models from PASCAL3D+, and also show that the adversarial behaviors of our "adversarial meshes" can transfer to black-box renderers. This provides us a better understanding of adversarial behaviors of 3D meshes in practice, and can motivate potential future defenses.

CHAPTER 4

Adversarial Examples in the Physical World

The previous chapter introduced the adversarial behavior of DNNs in the 3D space. In this chapter, we hope to demonstrate that an adversary could pose a threat in real-world complicated industry-level systems. Previous studies propose to generate physical attacks — stickers or printable textures that are able to fool a vision sensor on an autonomous driving vehicle to mis-recognize a stop sign in practice [5, 38]. However, real world systems, such as an autonomous driving system, would not only depend on vision sensors. Different sensor fusion mechanisms have been applied and demonstrated to help improve model robustness [75]. For instance, most autonomous driving detection systems are equipped with LiDAR (Light Detection And Ranging) or RADAR (Radio Detection and Ranging) devices which are able to directly probe the surrounding 3D environment with laser beams. As a result, the pure texture based perturbation would not be effective against such models given the fact that they cannot manipulate the object shape directly to mislead the beam reflection. Therefore, is it possible to generate "adversarial objects" in the 3D world to compromise LiDAR based recognition systems? Are these LiDAR based detection systems robust in practice? How effective these attack would be under various physical conditions?

Compared to the image based perturbation, there are several additional challenges for synthesizing adversarial objects against real-world LiDAR systems: 1) The LiDAR-based detection system consists of multiple non-differentiable components, rather than a single end-to-end network, which largely limits the use of existing gradient-based end-to-end attacks. 2) LiDAR-based detection system first projects 3D shape to a point cloud using physical LiDAR equipment, and then the point cloud is fed into the machine learning detection system. How the shape perturbation affects the scanned point clouds, as well as the projected 2D presentation is unclear. 3) The perturbation space against LiDAR systems is limited due to multiple aspects. First, we need to ensure the perturbed object can be reconstructed (3D printed) in real world and "looks like" a normal object, so it needs to be smooth and regular. Second, a valid LiDAR scan of an object is a constrained subset of point clouds, making the perturbation space much smaller compared to perturbing unconstrained point clouds or 2D image pixels directly [125].

To overcome the above challenges, we introduce LiDARadv to generate an adversarial object

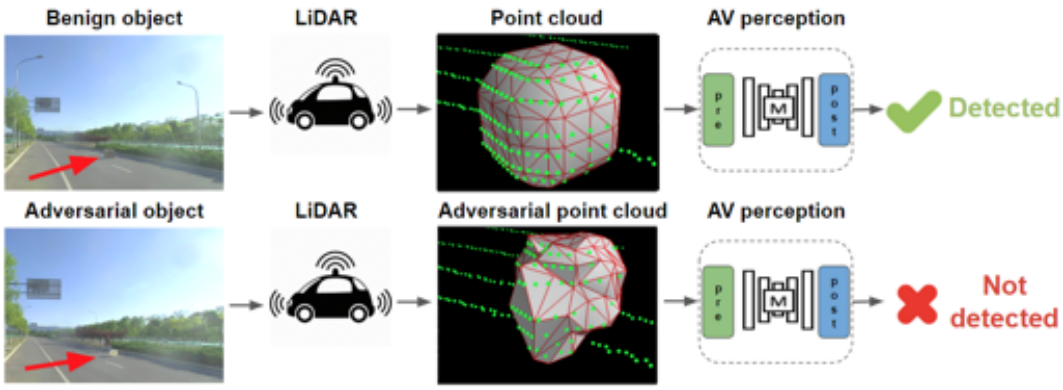


Figure 4.1: Overview of LiDARadv. The first row shows the control experiment where a regular box is detected by the LiDAR-based detection system; row 2 shows the generated adversarial object with similar size cannot be detected.

against the real-world LiDAR system The pipeline is shown in Figure 4.1. In detail, we first use a differentiable renderer to simulate the functionality of LiDAR. The differentiable design could make the flow between 3D objects and LiDAR scans (or point clouds) differentiable which further enables the end-to-end optimization. We then formulate 3D feature aggregation with a differentiable proxy function. Finally, we devise different loss functions to ensure the smoothness of the generated 3D adversarial objects. With such the system design, we could use the optimization-based method to generate adversarial examples.

To better demonstrate the flexibility of the proposed attack approach, we evaluate our attacking approach under two different attack scenarios: 1) *Hiding Object*: synthesizing an "adversarial object" that will not be detected by the detector. 2) *Changing Label*: synthesizing an "adversarial object" which will be mis-recognized as a specified adversarial target by the LiDAR detector.

To evaluate the real-world impact of LiDARadv, we 3D print out the generated adversarial objects and test them on the Baidu Apollo autonomous driving platform, an industry-level system which is not only widely adopted for research purpose but also actively used in industries. We show that based on the 3D perception LiDAR sensor and a production-level multi-stage detector, we are able to mislead the autonomous driving system to achieve different adversarial targets.

To summarize, our contributions in this chapter are as follows:

- We propose LiDARadv, an end-to-end approach to generate physically plausible adversarial objects against LiDAR-based autonomous driving detection systems.
- We select Baidu Apollo, an industry-level autonomous driving platform as the targeted system to illustrate the effectiveness and robustness of the attacks in practice. We also compare the objects generated by LiDARadv with an evolution algorithm to show that LiDARadv can provide smoother objects.

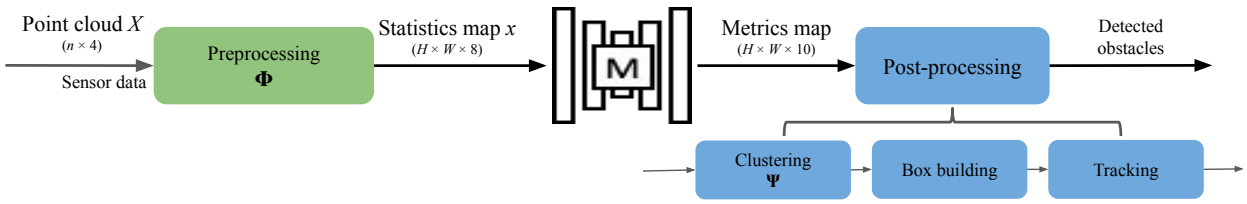


Figure 4.2: Overview of LiDAR-based detection on AV.

- We conduct physical experiments by 3D-printing the optimized adversarial object and show that it can consistently mislead the LiDAR system equipped in a moving car.

The organization of this chapter is as follows. Section 4.1 will give a system overview about LiDAR-based detection system of Baidu Apollo. Section 4.2 will introduce how to apply our universal formalization to solve this problem and the corresponding challenges and technique details. Section 4.3 will show the experimental results.

4.1 LiDAR-based Detection System

In this section, we provide the details of the LiDAR-based detection system that is directly related to our proposed adversarial attack An overview of the system is shown in Fig. 4.2. First, a LiDAR sensor scans the 3D environment and obtains a raw point cloud of the scene. Next, the point cloud goes through preprocessing, and is fed to a detection model. Finally, post-processing is applied to the detection output to obtain the detection predictions. **LiDAR** A LiDAR sensor scans the surrounding environment and generates a point cloud as $X \in \mathbb{R}^{n \times 4}$ (3D coordinates + intensity value). First, a sensor fires off an array of laser beams in horizontal and vertical directions. It then captures the light intensity reflected, and calculates the time that photons have traveled along each beam (Time of Flight). The distance and the coordinate of the surface points along the beam can then be computed, and these point clouds will form a raw point cloud of the object surfaces in the environment. LiDAR sensors are supposedly robust to object surface textures, as the Time of Flight is not easily affected by texture change. Though it also detects the intensity of reflected lights, it is unclear how adversarial algorithms designed for natural lighting in image space can be adapted to invisible laser beams used as light sources. Therefore, image-based adversarial attacks may have limited effects on such LiDAR-based detection system.

Preprocessing In this step, the input is the raw point cloud X from LiDAR, and the output is a statistics map of $H \times W \times 8$. First, the point cloud X is sliced into $H \times W$ vertical columns. For each column, the statistics of the points are then aggregated to form a vector of size 8, including intensity, point counts, average/max heights *etc*. (see Table. 4.1).

Table 4.1: Machine learning model input features extracted in the preprocessing phase.

Feature	Description
Max height	Maximum height of points in the cell.
Max intensity	Intensity of the highest point in the cell.
Mean height	Mean height of points in the cell.
Mean intensity	Mean intensity of points in the cell.
Count	Number of points in the cell.
Direction	Angle of the cells center with respect to the origin.
Distance	Distance between the cells center and the origin.
Non-empty	Binary value indicating whether the cell is empty or occupied.

Table 4.2: Output metrics of the segmentation model.

Metric	Description
Center offset (off)	Offset to predicted center of the cluster the cell belongs to.
Objectness (obj)	The probability of a cell belonging to an obstacle.
Positiveness (pos)	The confidence score of the detection.
Object height (hei)	The predicted object height.
i**-th Class Probability (cls**$_i$**)**	The probability of the cell being from class i (vehicle, pedestrian, etc.).

In this procedure, some dimensions (*e.g.* max height, count) introduce zero gradients due to the "hard" assignment of points to different columns (slicing), so the end-to-end optimization-based attack algorithms are not directly applicable.

Prediction (f) In this step, the input is the statistics map of size $H \times W \times 8$, and the output is a metrics map of $H \times W \times 10$. In Apollo, this is done by an autoencoder-like deep neural network. For the output metrics map of size $H \times W \times 10$, the 10-D detection vector in each column includes center offset (2D), objectness, positiveness, object height and 5 class probabilities. The descriptions of these metrics are detailed in Table. 4.2. This metric map is then used to determine the final detection result in the post-processing step.

Post-processing The post-processing phase aggregates previous prediction results from the deep networks and outputs the final object detection result. It can be roughly divided into 3 sequential components: *clustering*, *box building* and *tracking*. The clustering process composes obstacle candidates using both the metrics maps and the point cloud X. In the clustering process, *objectness* score, *center offset confidence* in the metrics maps are used to generate the initial candidates. The class probabilities in the metrics maps are then used to assign the object classes of the candidates. The box builder then reconstructs the bounding boxes including the height, width, length of the object candidates from the corresponding points. Finally, the tracker integrates multiple frames of

processed results to track the objects and to get additional information such as speed.

Note that in this chapter, we only consider a single frame for the adversarial attacks as a demonstration of feasibility. For the case of multiple frames, it can be treated as enhancing robustness against object motions, and such robustness against different locations is shown in experiments later (§ 4.3.4).

4.2 Generating Adversarial Object Against LiDAR-based Detection

4.2.1 Framework Overview

Given a 3D object S in a scene, a LiDAR sensor scans the scene to generate a point cloud X:

$$X = \text{render}(S, \text{background})$$

For pre-processing, this point cloud X is sliced and aggregated to generate the statistics map x of size $H \times W \times 8$:

$$x = \Phi(X)$$

Then a machine learning model M maps this 2D feature x to $O = f(x)$, where $O \in R^{H \times W \times 10}$ is the metrics map. O is then fed to the clustering process Ψ to generate the confidence y_{conf} and label y_{label} of detected obstacles: $(y_{\text{conf}}, y_{\text{label}}) = \Psi(O)$

An attacker aims to manipulate the object S to fool the detector. Here we define two types of adversarial goals: 1) *Hiding object*: Hide an existing object S by manipulating S; 2) *Changing label*: Change the label y of the detected object S to a specified target y'.

To achieve the above adversarial goals in LiDAR-based detection is non-trivial, and there are the following challenges: 1) **Multiple pre/post-processing stages.** Unlike the adversarial attacks on traditional image-space against machine learning tasks such as classification and object detection, the LiDAR-based detection here is not a single end-to-end learning model, It consists of the differentiable learning model M and several non-differentiable parts including preprocessing and post-processing. Thus, the direct gradient based attacks are not directly applicable. 2) **Manipulation constraints.** Instead of directly manipulating the point cloud X as in [125], we manipulate the 3D shape of S given the limitation of LiDAR. The points in X are the intersections of laser beams and object surfaces and cannot move freely, so the perturbations on each point may affect each other. Keeping the shape plausible and smooth adds additional constraints [129]. 3) **Limited Manipulation Space.** Consider the practical size of the object versus the size of the scene that is processed by LiDAR, the 3D manipulation space is rather small ($< 2\%$ in our experiments), as shown in Fig. 4.1.

Given the above challenges, we design an end-to-end attacking pipeline. We hope to change

the metrics map O, by alternating the input shape S, since O is a function of the input shape: $O = F(S) = f(\Phi(R(S)))$. However, the LiDAR scan $R(S)$ happens in the physics world and we can only simulate it; the function Φ has non-differentiable parts if we want to apply gradient-based optimizers. Now let's see how we deal with these problems.

4.2.2 Differentiable Renderer

LiDAR simulation The renderer R simulates the physics of a LiDAR sensor that probes the objects in the scene by casting laser beams. The renderer first takes a mesh S as input, and computes the intersections of a set of predefined ray directions to the meshes in the scene to generate point cloud X. After depth testing, the distance along each beam is then captured, representing the surface point of a mesh that it first encounters, as if a LiDAR system receives a reflection from an object surface. Knowing ray directions of the beams, the exact positions of the intersection points can be inferred from the distance, in the form of point clouds X.

In detail, the renderer simulates the physics of a LiDAR sensor that probes the objects in the scene by casting laser N_{ray} rays: $R = \{\mathbf{r}_i \in \mathbb{R}^3, \|\mathbf{r}_i\| = 1, i = 1, 2, \cdots, N_{\text{ray}}\}$, with $\mathbf{r}_i$ representing the direction of the i-th ray. Given a shape S with the surface ∂S as input, the renderer computes the intersections of rays R to the mesh faces in the scene. For each ray $\mathbf{r}_i$, the intersection coordinate P_i are computed through depth testing (assuming the center of the rays is at origin, *i.e.* the reference frame of LiDAR):

$$\mathbf{p}_i = \underset{\mathbf{p}}{\operatorname{argmin}}\{\|\mathbf{p}\| \mid \exists t > 0, \mathbf{p} = t \cdot \mathbf{r}_i, \mathbf{p} \in \partial S\}, \quad i = 1, 2, \cdots, N_{\text{ray}} \tag{4.1}$$

In detail, the shape S is described by triangle vertex coordinates and a fixed set of triangle faces. Then, this $\mathbf{p}_i$ is an intersection of the ray described by $\mathbf{r}_i$ and the plane described by the three vertex coordinates of the triangle it intersects, so $\mathbf{p}_i$ is a function differentiable to the triangle vertex coordinates in S.

Real background from a road scene We render our synthetic object onto a realistically captured point cloud. First, we obtain the 3D scan of a road scene, using the LiDAR sensor mounted on a car. Then, we obtain the laser beam directions by computing the normalized vectors from the origin (LiDAR) pointing to the scanned points. This fixed set of ray directions are then used for rendering our synthetic objects throughout the chapter.

Hybrid rendering of synthetic objects onto a realistic background Notice that we have a predefined set of rays R. To obtain these rays, one can refer to the specifications of a LiDAR device.

In our method, we directly compute the directions from the captured background point cloud P', so that the rays are close to real world cases:

$$\mathbf{r_i} = \frac{\mathbf{p}'_i}{\|\mathbf{p}'_i\|} \tag{4.2}$$

With this, Eq. (4.1) becomes:

$$\mathbf{p}_i = \underset{\mathbf{p}}{\operatorname{argmin}}\{\|\mathbf{p}\| \mid \mathbf{p} = \mathbf{p}'_i \vee \mathbf{p} = t \cdot \mathbf{r}_i, t > 0, \mathbf{p} \in \partial S\},\ i = 1, 2, \cdots, N_{\text{ray}} \tag{4.3}$$

This means when rays intersect with an object, the corresponding background points blocked by the above-ground parts of the object are removed during depth testing; if the object is below the ground, the intersections leave those corresponding background points intact also due to depth testing. In this way, we obtain a semi-real synthetic point cloud scene: the background points come from the captured real data; the foreground points are physically accurate simulations based on the captured real data.

4.2.3 Differentiable Proxy Function

As in Section 4.1, the computation of statistics map Φ includes operations such as **count**, **max height**, and **mean height**, **intensity** and **non-empty**: they are non-differentiable to the point coordinates because each point is assigned to one column grid. If we want to apply end-to-end optimizers to for our synthetic object S, we need to pass the gradient through this step, with the help of our proxy functions.

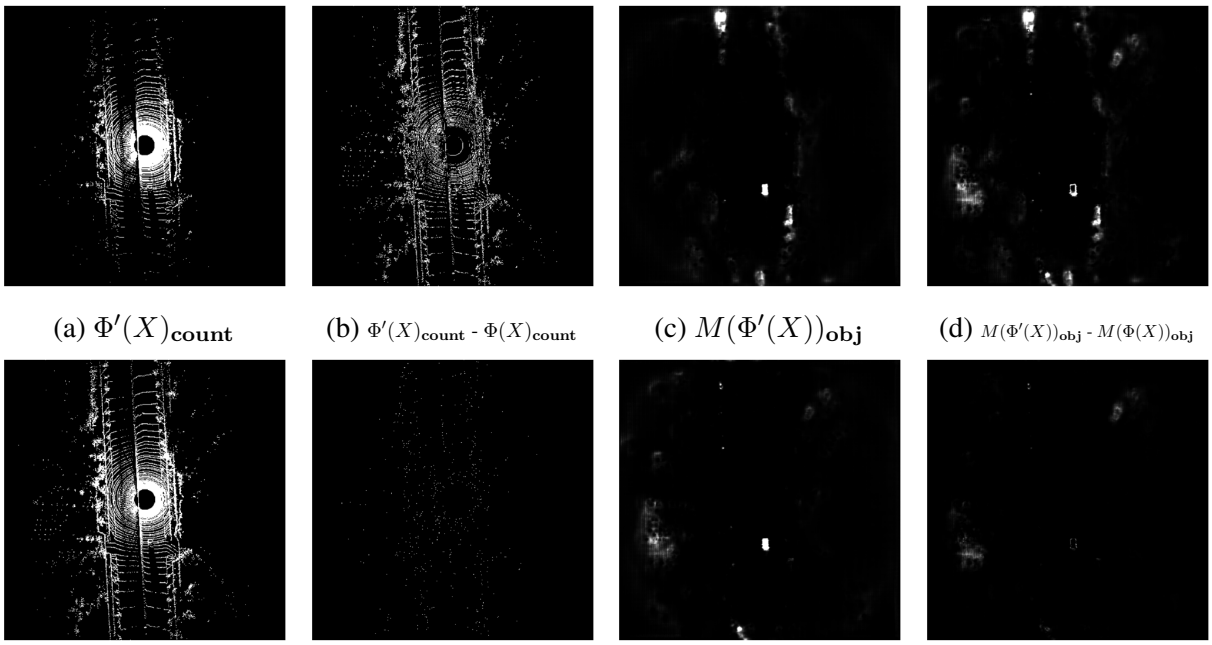
Given a point cloud X with coordinate (u^X, v^X, w^X), we count the number of points falling into the cells of a 3D grid $G \in \mathbb{R}^{H \times W \times P}$. For a point X_i with location$(u^{X_i}, v^{X_i}, w^{X_i})$, it is assigned to cells, with trilinear weights based on the distance to the cell centers:

$$G_{c,i} = (1 - d(u_c, u^{X_i})) \cdot (1 - d(v_c, w^{X_i})) \cdot (1 - d(w_c, w^{X_i})), \tag{4.4}$$

where j is the cell index and i is the point index and $d(\cdot, \cdot)$ represents the L_1 distance with a max of 1. $1 - d(\cdot, \cdot)$ means how much the point i contributes to the statistics in cell c. Note that at most 8 cells can have non-zero weights for any point i, so G is a sparse matrix.

Then, the **count** statistics $x_{\mathbf{count}}$ for cell c is the value $G_c = \sum_i G_{c,i}$. Note that this count is no longer an integer and can have non-zero gradients w.r.t. the point coordinates.

We then use this soft count G_c to compute "mean height" and "max height" statistics. Assume T_c stores the height of cell c. Next, we can formulate the **mean height** $x_{\mathbf{mean-height}}$ and **max**



(a) $\Phi'(X)_{\mathbf{count}}$ (b) $\Phi'(X)_{\mathbf{count}}$ - $\Phi(X)_{\mathbf{count}}$ (c) $M(\Phi'(X))_{\mathbf{obj}}$ (d) $M(\Phi'(X))_{\mathbf{obj}}$ - $M(\Phi(X))_{\mathbf{obj}}$

(e) $\Phi''(X)_{\mathbf{count}}$ (f) $\Phi''(X)_{\mathbf{count}}$ - $\Phi(X)_{\mathbf{count}}$ (g) $M(\Phi''(X))_{\mathbf{obj}}$ (h) $M(\Phi''(X))_{\mathbf{obj}}$-$M(\Phi(X))_{\mathbf{obj}}$

Figure 4.3: The performance of trilinear approximator and tanh approximator. The format "$\phi(\cdot)''_{\mathbf{count}}$ represents the 2D count feature calculated by trilinear approximator Φ'; $M(\Phi'(X))_{\mathbf{obj}}$ represents visualization of the "objectiveness " metric in the output of model M using trilinear approximator with; $\Phi'(X)_{\mathbf{count}}$ - $\Phi(X)_{\mathbf{count}}$ represents the approximator 's error of Φ'. The same notation for tanh approximiator Φ''

height $x_{\mathbf{max-height}}$ by weighting T_c with soft counts:

$$x_{\mathbf{mean-height}} = \frac{\sum_{c\in C} G_c \cdot T_c}{\sum_{c\in C} G_c + \epsilon} \quad \text{and} \quad x_{\mathbf{max-height}} = \max_{c\in C} \operatorname{sign}(G_c) \cdot T_c \tag{4.5}$$

where C is the set of all the cells in a column which we want to comput the statistics for. ϵ is to prevent zero denominators. Note that the sign function (indicating whether there are points in the cell) is non-differentiable, so we approximate the gradient using $\operatorname{sign}(G) = G$ during back propagation. The feature **intensity** has the similar formulation of **height** so we omit them here. The feature **non-empty**, formulated as $x_{\mathbf{non-empty}} = \operatorname{sign}(G)$, is also dealt in this way.

We denote the above trilinear approximator as Φ', in constrast to the original non-differentiable preprocessing step Φ. A visualization of output of our $\Phi'(X)_{\mathbf{count}}$ compared to the original $\Phi(X)_{\mathbf{count}}$ is shown in Figure 4.3. Since our approximation introduces differences in counting, $\Phi'(X)$ is not strictly equal to $\Phi(X)$, resulting in different obj (objectiveness) values of the final model prediction. We observe that this difference will raise new challenges to transfer the adversarial object generated based on Φ' to Φ. To solve this problem, we reduce the difference between Φ' and Φ, by replacing the L1 distance d in Eq. 4.4 with $d(u_1, u_2) = 0.5 + 0.5 \cdot \tanh(\mu \cdot (u_1 - u_2 - 1))$

where $\mu = 20$. We name this approximation "tanh approximator" and denote it Φ''. We observe that the input difference between Φ'' and the original Φ is largely reduced compared to Φ', allowing for smaller errors of the model predictions and better transferability. To extend our approximator and further reduce the gap betweer Φ'' and Φ, we interpolate the distance: $d(u_1, u_2) = \alpha \cdot (0.5 + 0.5 \cdot \tanh(5\mu \cdot (u_1 - u_2 - 1))) + (1 - \alpha) \cdot (u_1 - \lfloor u_2 \rfloor)$, where α is a hyper-parameter balancing the accuracy of approximation and the availability of gradients.

4.2.4 Objective Functions

Our objective is to generate a synthetic adversarial object S_{adv} from an original object S by perturbing its vertices, such that the LiDAR-based detection model will make incorrect predictions. We first optimize S_{adv} against the semi-real simulator detection model f.

$$\mathcal{L}(S_{\text{adv}}) = \mathcal{L}_{\text{adv}}(S_{\text{adv}}, f) + \lambda \mathcal{L}_{perceptual}(S_{\text{adv}}; S)\ . \tag{4.6}$$

The objective function $\mathcal{L}$ consists of two losses. $\mathcal{L}_{\text{adv}}$ is the adversarial loss to achieve the target goals while the $\mathcal{L}_{perceptual}$ is the distance loss to keep the properties of the "realistic" adversarial 3D object S_{adv}. We optimize the objective function by manipulating the vertices. The distance loss is defined as follows:

$$\mathcal{L}_{perceptual} = \sum_{\boldsymbol{v}_i \in V} \sum_{q \in \text{neighbor}(\boldsymbol{v}_i)} \|\Delta \boldsymbol{v}_i - \Delta \boldsymbol{v}_q\|_2^2 + \beta \sum_{\boldsymbol{v}_i \in V} \|\Delta \boldsymbol{v}_i\|_2^2\,, \tag{4.7}$$

where $\Delta \boldsymbol{v}_i = \boldsymbol{v}_i^{\text{adv}} - \boldsymbol{v}_i$ represents the displacement between the adversarial vertex $\boldsymbol{v}_i^{\text{adv}}$ and pristine vertex $\boldsymbol{v}_i$. β is the hyperparameter balancing these two losses. The first losses [129] is a Laplacian loss preserving the smoothness of the perterbation applied on the adversarial object S_{adv}. The second part is the $L2$ distance loss to limit the magnitude of perturbation.

Objective: hide the inserted adversarial object As introduced in the background section, the existence of the object highly depends on the "positiveness" metric, namely the confidence. $H(*, M, S)$ denotes a function extracting $*$ metric from the model M given an object S. $\mathcal{A}$ is the mask of the target object's bounding box. Our adversarial loss is represented as follows:

$$\mathcal{L}_{\text{adv}} = H(\text{pos}, M, S) \cdot \mathcal{A} \tag{4.8}$$

Objective: changing label In order to change the predicted label of the object, it needs to increase the logits of the target label and decrease the logits of the ground-truth label. Moreover, it also

needs to preserve the high positiveness. Based on this, our adversarial loss is written as

$$\mathcal{L}_{\text{adv}} = (-H(\mathrm{c}_{y'}, M, S) + H((c)_y, M, S) \cdot \mathcal{A} \cdot H(\text{pos}, M, S) \tag{4.9}$$

In order to ensure that adversarial behaviors still exist when the settings are slightly different, we create robust adversarial objects that can perform successful attacks within a range of settings, such as different object orientations, different positions to the LiDAR sensor *etc*. To achieve such goal, we sample a set of physical transformations to optimize the loss expectation. In reality, we create a victim set D by rendering the object S at different positions and orientations. Instead of optimizing an adversarial object S by attacking single position and orientation, we generate a universal adversarial object S to attack all positions and orientations in the victim set D.

4.2.5 Blackbox Attack

In reality, it is possible that the attackers do not have complete access to the internal model parameters, *i.e*. the model is a black box. Therefore, in this subsection, we also develop an evolution-based approach to perform blackbox attack.

In evolution, a set of individuals represent the solutions in the search space, and the fitness score defines how good the individuals are. In our case, the individuals are mesh vertices of our adversarial object, and the fitness score is $-\mathcal{L}(S_{\text{adv}})$. We initialize m mesh vertices using the benign object S. For each iteration, new population of n mesh vertices is generated by adding random perturbations, drawn from a Gaussian distribution $\mathcal{N}(0, \sigma)$, to each mesh vertices in the old population. m mesh vertices with top fitness scores will remain for the next iteration, while the others will be replaced. We iterate the process until we find a valid solution or reach a maximum number of steps.

4.3 Experiments

In this section, we first expose the vulnerability of the LiDAR-based detection system via the evolution-based blackbox algorithm by achieving the goal of "hiding object", because missing obstacles can cause accidents in real life. We then show the qualitative results and quantitative results of LiDARadv under whitebox settings. In addition, we also show that LiDARadv can achieve another adversarial goals such as "changing label". Moreover, the point clouds are continuously captured in real life, so attacks in a single static frame may not have much effect in real-world cases. Therefore, in our experiments, we generate a universal robust adversarial object against a victim dataset which consists of different orientations and positions. We 3D-print such universal adversarial object and conduct the real-world drive-by experiments, to show that they indeed can pose a threat on road.

4.3.1 Experimental Setup

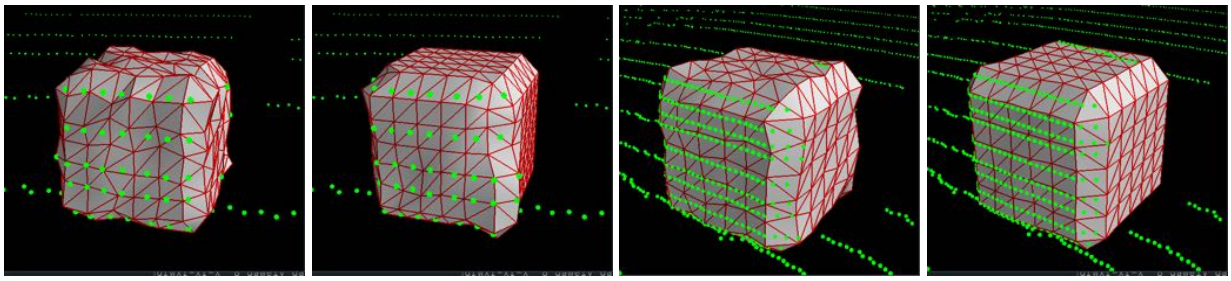
Autonomous Driving Systems. We evaluate on the perception module of Baidu Apollo, an open source autonomous driving platform.

LiDARadv Settings We initialize the adversarial object as a resampled 3D cube-shaped CAD model using MeshLab [25]. For rendering, we implement a fully differential LiDAR simulator with predefined laser beam ray directions extracted from a real scene captured by the Velodyne HDL-64E sensor, as stated in § 4.2.2. It has around 2000 angles in the azimuth angle and around 60 angles in the elevation angle. We use Adam [67], and choose λ as 0.003 in Eq. 4.6 using binary search. For the evolution-based blackbox algorithm , we choose $\sigma = 0.1$, $n = 500$ and $m = 5$.

4.3.2 LiDARadv under Blackbox Settings

We first evaluate the existence of the vulnerability using our evolution-based blackbox attacks, with the goal of "hiding object". We generate adversarial objects in different size (50cm and 75cm in edge length). For each object, we select 45 different position and orientation pairs for evaluation, and the results are shown in Table 4.3. The results indicate that the LiDAR-based detection system is vulnerable. The visualization of the adversarial object is shown in Figure 4.4(a) and (c).

4.3.3 LiDARadv with Different Adversarial Goals



(a) 50cm: Evo. (b) LiDARadv (c) 75cm: Evo. (d) LiDARadv

Figure 4.4: Adversarial meshes of different sizes can fool the detectors even with more LiDAR hits. We generate the object with LiDARadv and evolution-based method (Evo.).

After showing the vulnerability of the LiDAR-based detection system, here we focus on whitebox settings to explore what a powerful adversary can do, since "the design of a system should not require secrecy" [103]. Therefore, we evaluate the effectiveness of our whitebox attack LiDARadv with the goal of "hidding object". We also evaluate the feasibility of LiDARadv to achieve another goal of "changing label".

Table 4.3: Attack success rate of LiDARadv and evolution based method under different settings.

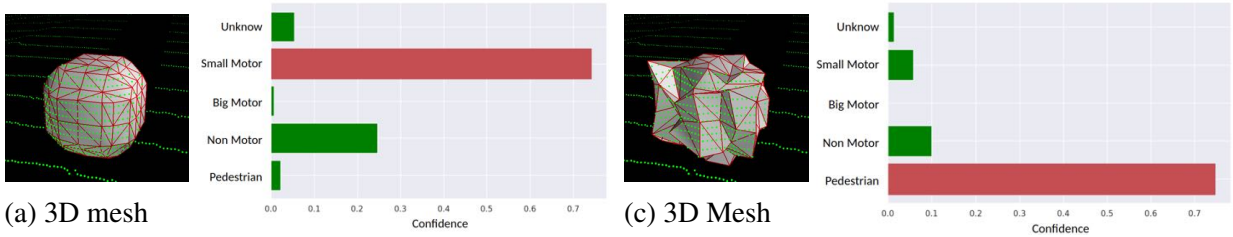
Attacks	Object size	
	50cm	75cm
LiDARadv	32/45 (71%)	23/45 (51%)
Evolution-based	28/45 (62%)	16/45 (36%)

Hiding object We follow the same settings as in the above sections, and Table 4.3 shows the results. We find that LiDARadv can achieve 71% attack success rate with size 50cm. The attack success rate is consistently higher than the evolution-based blackbox attacks. Figure 4.4 (b) and (c) show the visualizations of the adversarial objects. We visually observe that the adversarial objects generated by LiDARadv are smoother than that of evolution.

Table 4.4: The attack success rate of the adversarial objects generating using LiDARadv, starting from different types of pristine meshes. The target labels are the other four labels different from the original predictions.

	Cube	Sphere	Tetrahedron	Cylinder	Overall
Attack Success Rate	100%	100%	75%	50%	75%

Changing label The result shown in Figure 4.5 and Table 4.4 indicates that we can successfully change the label of the object. We also experiment with different initial shapes and target labels. We select 3 pristine meshes (cuba, sphere, tetrahedron) and set the target label to the other 4 labels except for the original label. The results are shown in Table 4.4, showing that our LiDARadv has a high chance to trick the detector to output target labels, regardless of different pristine meshes that it starts from.



(a) 3D mesh (Benign)

(b) Predictions (Benign)

(c) 3D Mesh (Adv.)

(d) Predictions (Adv.)

Figure 4.5: The adversarial mesh generated by LiDARadv is mis-detected as a “Pedestrian”.

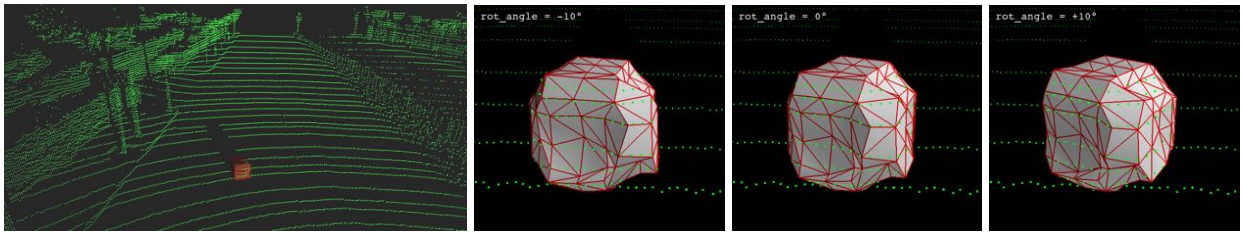
Table 4.5: Robust Adversarial Object against different angles. The original confidence is x. Our success rate is 100%. (✓represents no object detected)

Angle		-10°	-5°	0°	5°	10°
Objectness (Confid.)	Model	✓	✓	✓	✓	✓
	Apollo	✓	✓	✓	✓	✓

4.3.4 LiDARadv on Generating Robust Physical Adversarial Objects

To ensure the generated LiDARadv preserves adversarial behaviors under various physical conditions, we optimize the object by sampling a set of physical transformations such as possible positions and orientations as follows.

Before we 3D-print the object, we add additional results to evaluate the robustness of the generated objects against different positions and different angles. By doing so, it can provide insight into the performance of our adversarial object in real-world settings.



(a) Benign frame (b) Adv (-10°) (c) Adv (0°) (d) Adv (10°)

Figure 4.6: The visualization of the adversarial object with different angles. In the benign frame (a), the system is able to detect the cube. When we replace the cube with our adversarial object, the system fails to detect the object at all three angles. We visualize the mesh along with the point clouds in a close-up view in (b), (c) and (d).

Different Angles We generate the adversarial objects by attacking for 9 angles simultaneously and evaluate the attack success rate among these angles. Our approach achieves 100% attack success rate (Table 4.5) both on our approximate model and the Apollo system. This indicates that our designed differentiable proxy functions are accurate enough to transfer the adversarial behavior to Apollo. Figure 4.6 shows qualitative results of the adversarial object from different close-up views. We can observe that the adversarial example is smooth and can be easily reconstructed in the real-world.

Different Positions Similarly, we generate a single robust adversarial object against different positions simultaneously, as is shown in Figure 4.7. We select 9 positions and use our algorithm

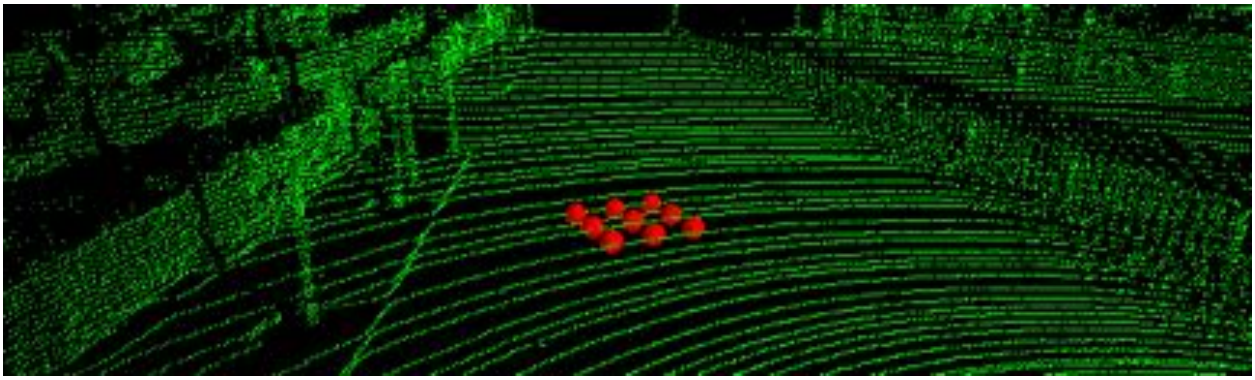


Figure 4.7: Our adversarial object can successfully attack the detection system, while placed at different positions. The red spheres mark the locations we place the adversarial object.

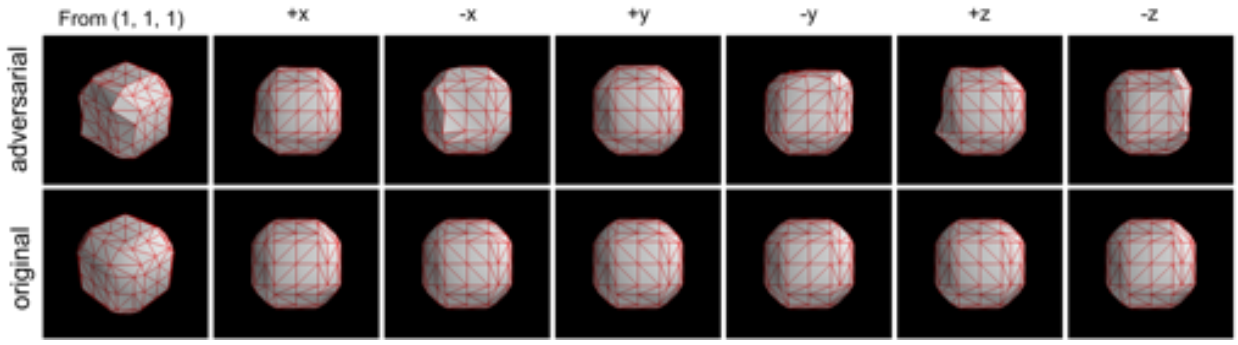


Figure 4.8: The optimized robust adversarial objects from 6 principal views and a particular view, compared with the original pristine object.

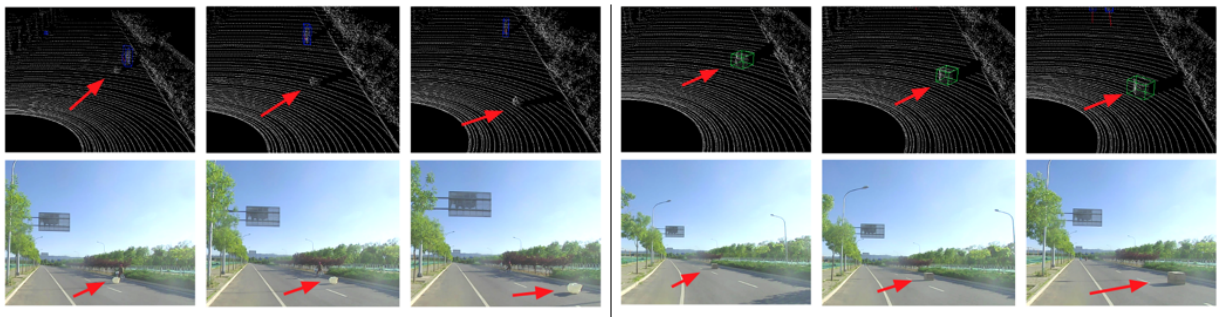
Table 4.6: Robust Adversarial Object against different positions. The original object can be detected by Apollo. Our success rate is 100%. (✓represents no object detected)

Position	Objectness (Confid.)		Position	Objectness (Confid.)		Position	Objectness (Confid.)	
	Ours	Apollo		Ours	Apollo		Ours	Apollo
(-50, -50)	✓	✓	(0, -50)	✓	✓	(50, -50)	✓	✓
(-50, 0)	✓	✓	(0, 0)	✓	✓	(50, -50)	✓	✓
(-50, 50)	✓	✓	(0, 50)	✓	✓	(50, 50)	✓	✓

to generate a universal robust adversarial example against different positions. Figure 4.8 shows 7 views of the generated object from different angles, compared to the original object. This adversarial example is smooth from all views. It shows that our approach is able to achieve the goal while keeping the shape plausible, so we can easily print the object to perform the physical attack. Table 4.6 show the detailed results of our adversarial object against these 9 positions: it can successfully attack the system among these 9 positions. We show that the generated robust adversarial object is able to achieve the attack goal of hiding the object with a high success rate in Table 4.7. An interesting phenomenon is that some attack performance under the unseen settings is

Table 4.7: Attack success rates of LiDARadv at different positions and orientations under both controlled and unseen settings.

Controlled Setting		Unseen Setting			
		Distance (cm)		Orientation (°)	
Distance (cm) & Orientation (°)	Attack	0-50	50-100	0-5	0-10
$\{0, \pm 50\} \times \{0, \pm 2.5, \pm 5\}$	91%	96%	91%	100%	90%
$\{0, \pm 50\} \times \{0, \pm 2.5, \pm 5\}$	96%	96%	90%	80%	100%



(a) Adversarial object

(b) Benign object

Figure 4.9: Results of physical attack. Our 3D-printed robust adversarial object by LiDARadv is not detected by the LiDAR-based detection system in a moving car. Row 1 shows the point cloud data collected by LiDAR sensor, and Row 2 presents the corresponding images captured by a dash camera.

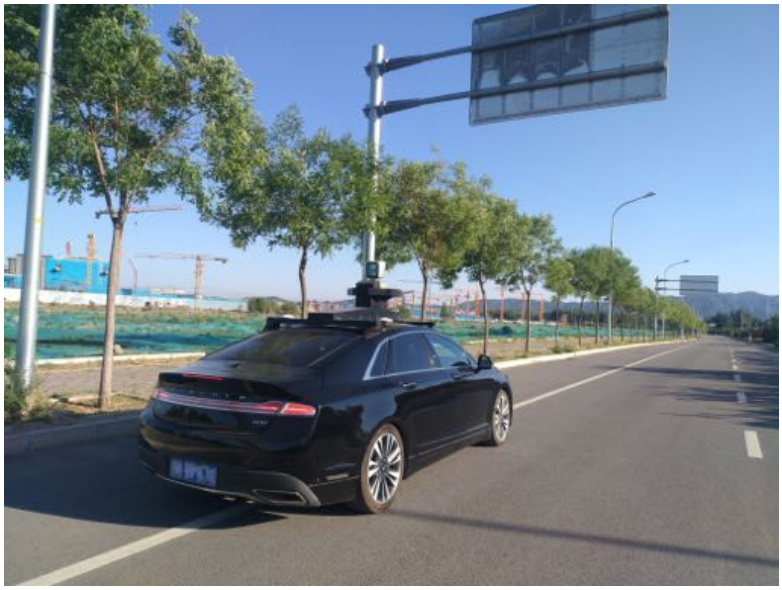
even better than that within the controlled environment. This implies that our adversarial objects are robust enough to generalize to unseen settings.

Physical Furthermore, we evaluate the generated robust adversarial object in the physical world by 3D printing the generated object. We 3D-print our robust adversarial object at 1:1, and drive a real car mounted with LiDAR and dashcams. The adversarial object is put on the road, and a car drives by, collecting scanned point clouds and the reference dashcam videos. For comparison, we also put the benign object, which is a box of same size at the same location and follow the same protocol when collecting the point clouds. Figure 4.10 shows our physical experiment setting.

We collect the point cloud data using a Velodyne HDL-64E sensor with a real car driving by and evaluate the collected traces on the LiDAR perceptual module of Baidu Apollo. As shown in Figure 4.9a, we find that the adversarial object is not detected around the target position among all 36 different frames. To compare, the box object (in Figure 4.9b) is detected in 12 frames among all 18 frames. The number of total frames is different due to the different vehicle speeds.

(a) the road where we perform the physical experiment

(b) the benign object for comparison



(c) the car used to collect the dashcam videos and the point clouds

(d) our adversarial object

Figure 4.10: Our physical experiment setting. We 3D-print the generated adversarial object at 1:1, and drive a car mounted with LiDAR and dashcams to collect the scanned point clouds and the reference videos.

4.4 Conclusions

In this chapter, we have showed that LiDAR-based detection systems for autonomous driving are vulnerable against adversarial attacks. By integrating our proxy differentiable approximator, we are able to generate robust physical adversarial objects. We show that the adversarial objects are able to attack the Baidu Apollo system at different positions with various orientations. We also

show LiDARadv can generate a much smoother object than the evolution based attack algorithm. Our findings raise great concerns about the security of LiDAR systems in AV, and we hope this work will shed light on potential defense methods.

CHAPTER 5

Detecting Adversarial Examples by Using the Property of the Learning Model

Previous chapters have introduced the vulnerability of DNNs to *adversarial examples*. In the following chapters, we discuss the defense. Here, we aim to defend against adversarial examples in the digital space and leave the physically adversarial examples as the future work.

Given the intriguing properties of adversarial examples, various analyses for understanding adversarial examples have been proposed [84, 85, 121, 122], and potential defense/detection techniques have also been discussed mainly for the image classification problem [30, 53, 85]. For instance, image pre-processing [36], adding another type of random noise to the inputs [133], and adversarial retraining [41] have been proposed for defending/detecting adversarial examples when classifying images. However, researchers [17, 49] have shown that these defense or detection methods are easily attacked again by attackers with or even without knowledge of the defender's strategy. Such observations bring up concerns about safety problems within diverse machine learning based systems.

In order to better understand adversarial examples against different tasks, in this chapter, we aim to analyze adversarial examples in the semantic segmentation task instead of classification. We hypothesize that adversarial examples in different tasks may contain unique properties that provide in-depth understanding of such examples and encourage potential defensive mechanisms. Different from image classification, in semantic segmentation, each pixel will be given a prediction label which is based on its surrounding information [29]. Such spatial context information plays a more important role for segmentation algorithms, such as [69, 76, 137, 148]. Whether adversarial perturbation would break such spatial context is unknown to the community. In this chapter, we propose and conduct image spatial consistency analysis, which randomly selects overlapping patches from a given image and checks how consistent the segmentation results are for the overlapping regions. Our pipeline of spatial consistency analysis for adversarial/benign instances is shown in Figure 5.1. We find that in segmentation task, adversarial perturbations can be weakened for separately selected patches, and therefore adversarial and benign images will show very different behaviors in terms of the spatial consistency information. Moreover, since such spatial consistency

is highly random, it is hard for adversaries to take such constraints into account when performing adaptive attacks. This renders the system less brittle even facing the sophisticated adversaries, who have full knowledge about the model as well as the detection/defense method applied..

We use image scale transformation to perform detection of adversarial examples as a baseline, which has been used for detection in classification tasks [111]. We show that by randomly scaling the images, adversarial perturbation can be destroyed and therefore adversarial examples can be detected. However, when the attacker knows the detection strategy (adaptive attacker), even without the exact knowledge about the scaling rate, attacker can still perform adaptive attacks against the detection mechanism, which is similar to the findings in classification tasks [17]. On the other hand, we show that by incorporating spatial consistency check, existing semantic segmentation networks can detect adversarial examples (average AUC 100%), which are generated by the state-of-the-art attacks considered in this chapter, regardless of whether the adversary knows the detection method. Here, we allow the adversaries to have full access to the model and any detection method applied to analyze the robustness of the model against adaptive attacks. We additionally analyze the defense in a black-box setting, which is more practical in real-world systems.

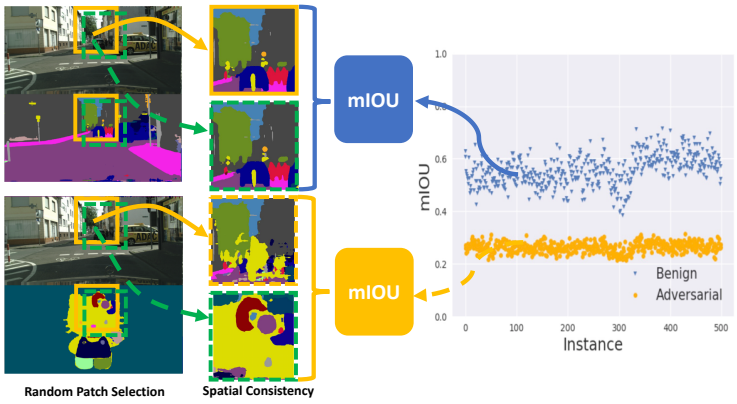


Figure 5.1: Spatial consistency analysis for adversarial and benign instances in semantic segmentation.

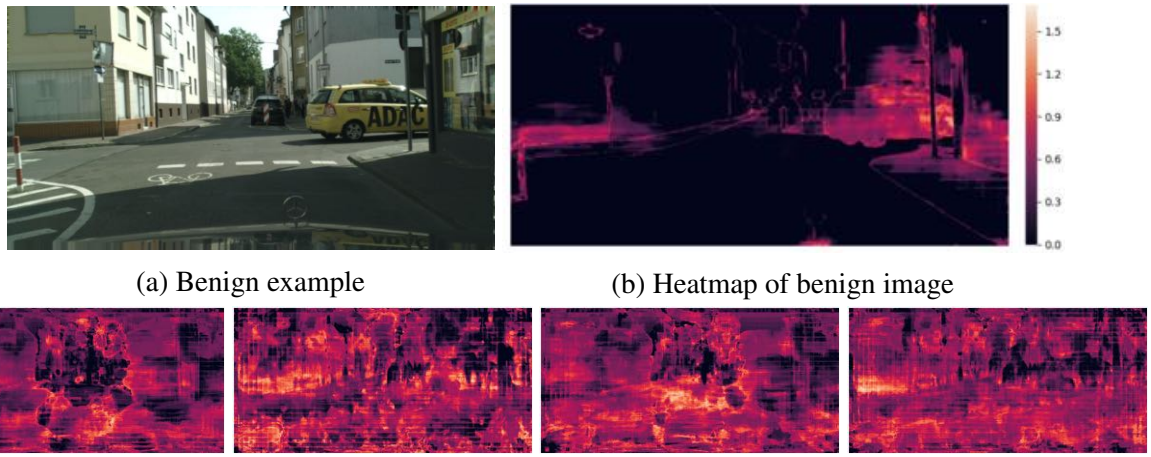
5.1 Spatial Consistency Based Method

In this section, we will explore the effects that spatial context information has on benign and adversarial examples in segmentation models. We conduct different experiments based on various models and datasets, and due to the space limitation, we will use a small set of examples to demonstrate our discoveries and relegate other examples to the appendix. Figure 5.2 shows the benign and adversarial examples targeting diverse adversarial targets: “Hello Kitty” (Kitty) and random pure color (Pure) on Cityscapes; and “Hello Kitty” (Kitty) and a real scene without any

(a) Cityscapes (b) BDD

Figure 5.2: Samples of benign and adversarial examples generated by Houdini on Cityscapes (targeting on Kitty/Pure) and BDD100K (targeting on Kitty/Scene). We select DRN as our target model here. Within each subfigure, the first column shows benign images and corresponding segmentation results, and the second and third columns show adversarial examples with different adversarial targets.

cars (Scene) on BDD video dataset, respectively. In the rest of the chapter, we will use the format "attack method — target" to label each adversarial example. Here we consider both DAG [132] and Houdini [27] attack methods.



(a) Benign example (b) Heatmap of benign image

(c) DAG — Kitty (d) DAG — Pure (e) Houdini — Kitty (f) Houdini — Pure

Figure 5.3: Heatmap of per-pixel self-entropy on Cityscapes dataset against DRN model. (a) and (b) show a benign image and its corresponding per-pixel self-entropy heatmap. (c)-(f) show the heatmaps of the adversarial examples generated by DAG and Houdini attacks targeting "Hello Kitty" (Kitty) and random pure color (Pure).

5.1.1 Spatial Context Analysis

To quantitatively analyze the contribution of spatial context information to the segmentation task, we first evaluate the entropy of prediction based on different spatial contexts. For each pixel m within an image, we randomly select K patches $\{P_1, P_2, ..., P_K\}$ which contain m. Afterward, within each patch P_i, the pixel m will be assigned with a confidence vector based on Softmax

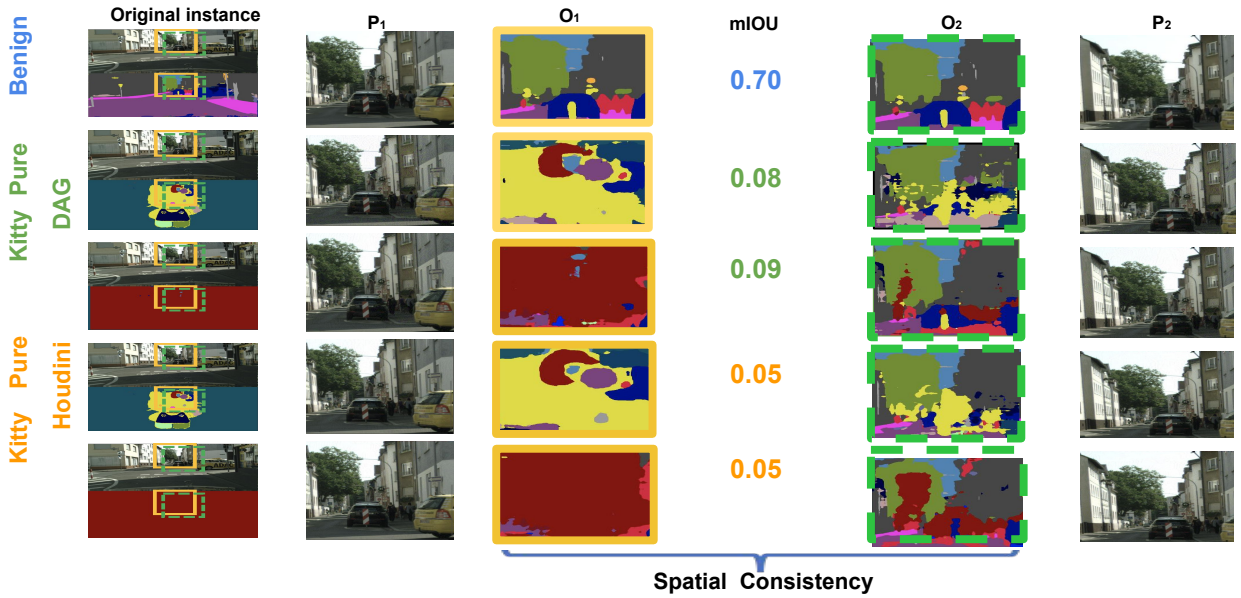


Figure 5.4: Examples of spatial consistency based method on adversarial examples generated by DAG and Houdini attacks targeting on Kitty and Pure. First column shows the original image and corresponding segmentation results. Column P_1 and P_2 show two randomly selected patches, while column O_1 and O_2 represent the segmentation results of the overlapping regions from these two patches, respectively. The mIOU between O_1 and O_2 are reported. It is clear that the segmentation results of the overlapping regions from two random patches are very different for adversarial images (low mIOU), but relatively consistent for benign instance (high mIOU).

prediction, so pixel m will correspond to K vectors in total. We discretize each vector to a one-hot vector and sum up these K one-hot vectors to obtain vector $\mathcal{V}_m$. Each component $\mathcal{V}_m[j]$ of the vector represents the number of times pixel m is predicted to be class j. We then normalize $\mathcal{V}_m$ by dividing K. Finally, for each pixel m, we calculate its self-entropy

$$\mathcal{H}(m) = -\sum_j \mathcal{V}_m[j] \log \mathcal{V}_m[j]$$

and therefore calculate the self entropy for each vector. We utilize such entropy information of each pixel to convey the consistency of different surrounding patches and plot this information in the heatmaps in Figure 5.3. It is clear that for benign instances, the boundaries of original objects have higher entropy, indicating that these are places harder to predict and can gain more information by considering different surrounding spatial context information.

5.1.2 Patch Based Spatial Consistency

The fact that surrounding spatial context information shows different spatial consistency behaviors for benign and adversarial examples motivates us to perform the spatial consistency check

hoping to potentially tell these two data distributions apart.

First, we introduce how to generate overlapping spatial contexts by selecting random patches and then validate the spatial consistency information. Let s be the patch size and w, h be the width and height of an image x. We define the first and second patch based on the coordinates of their top-left and bottom-right vertices $(u_1, u_2, u_3, u_4), (v_1, v_2, v_3, v_4)$, where Let $(d_{u_1,v_1}, d_{u_2,v_2})$ be displacement between the top-left coordinate of the first and second patch: $d_{u_1,v_1} = v_1 - u_1, d_{u_2,v_2} = v_2 - u_2$. To guarantee that there is enough overlap, we require (d_{u_1,v_1} and d_{u_2,v_2}) to be in the range $(b_{\mathbf{low}}, b_{\mathbf{upper}})$. Here we randomly select the two patches, aiming to capture diverse enough surrounding spatial context, including information both near and far from the target pixel. The **patch selection algorithm (getOverlapPatches)** is shown in Algo 1.

Algorithm 1: Patch Selection Algorithm ($\mathbf{getOverlapPatches}$)

input : patch size s
image width w
image height h
bound $b_{\mathbf{low}}, b_{\mathbf{upper}}$

output : Two random patches P_1 and P_2: $(u_1, u_2, u_3, u_4), (v_1, v_2, v_3, v_4)$

Generate two random integer numbers I_1, I_2, where $0 < I_1 < w - s - b_{\mathbf{upper}}, 0 < I_2 < h - s - b_{\mathbf{upper}}$. ;

Generate two random integer numbers I_3, I_4, where $b_{\mathbf{low}} < I_3, I_4 < b_{\mathbf{upper}}$. ;

Return: $(I_1, I_2, I_1 + s, I_2 + s), (I_1 + I_3, I_2 + I_4, I_1 + I_3 + s, I_2 + I_4 + s)$

Next we show how to apply the spatial consistency based method to a given input and therefore recognize adversarial examples. The detailed algorithm is shown in Algorithm 2. Here K denotes the number of overlapping regions for which we will check the spatial consistency. We use the mean Intersection Over Union (mIOU) between the overlapping regions O_1, O_2 from two patches P_1, P_2 to measure their spatial consistency. The mIOU is defined as $\frac{1}{n_{cls}} \sum_i n_{ii} / (\sum_j n_{ij} + \sum_j n_{ji} - n_{ii})$, where n_{ij} denotes the number of pixels predicted to be class i in O_1 and class j in O_2, and n_{cls} is the number of the unique classes appearing in both O_1 and O_2. $\mathbf{getmIOU}$ is a function that computes the mIOU given patches P_1, P_2 along with their overlapping regions O_1 and O_2.

5.2 Scale Consistency Analysis

We have discussed how spatial consistency can be utilized to potentially characterize adversarial examples in segmentation task. In this section, we will discuss another baseline method: image scale transformation, which is another natural factor considered in semantic segmentation [62, 81]. Here we focus on image blur operation by applying Gaussian blur to given images [19], which is studied for detecting adversarial examples in image classification [111]. Similarly, we

Algorithm 2: Spatial Consistency Check Algorithm

input: Input image x;
number of overlapping regions K;
patch size s;
segmentation model f;
bound $b_{\mathbf{low}}, b_{\mathbf{upper}}$;
output: Spatial consistency threshold c;

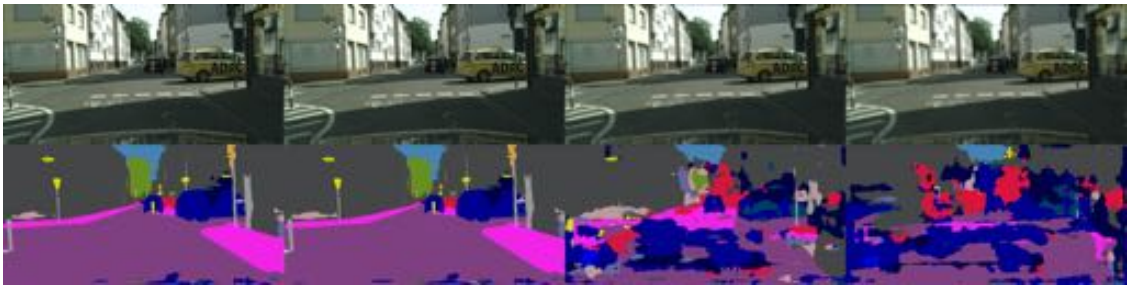
Initialization : **cs** $\leftarrow$[], $w \leftarrow x.width, h \leftarrow x.height$;
for $k \leftarrow 0$ **to** K **do**
  $(u_1, u_2, u_3, u_4), (v_1, v_2, v_3, v_4) \leftarrow \mathbf{getOverlapPatches}(s, w, h, b_{\mathbf{low}}, b_{\mathbf{upper}})$;
  $P_1 = X[u1 : u3, u2 : u4], P_2 = X[v_1 : v_3, v_3 : v_4]$;
  /* get prediction result of two random patches from f */;
  $\mathit{pred}^1 \leftarrow \text{argmax}_c\, f_c(P_1), \mathit{pred}^2 \leftarrow \text{argmax}_c\, f_c(P_2)$;
  /* get prediction of the overlap area between two patches */;
  $p_1 \leftarrow \{\mathit{pred}^1_{i,j} | \forall (i,j) \in \mathit{pred}^1, i > v_1 - u_1, j > v_2 - u_2\}$;
  $p_2 \leftarrow \{\mathit{pred}^2_{i,j} | \forall (i,j) \in \mathit{pred}^2, i < s - (v_1 - u_1), j < s - (v_2 - u_2)\}$;
  /* get consistency value (mIOU) from two patches */;
  cs $\xleftarrow{+}$ $\mathbf{getmIOU}(p1, p2)$;
end
$c \leftarrow \mathbf{Mean}(\mathbf{cs})$;
Return: c

will analyze the effects of image scaling on benign/adversarial samples. Since spatial context information is important for segmentation task, scaling or performing segmentation on small patches may damage the global information and therefore affect the final prediction. Here we aim to provide quantitative results to understand and explore how image scale transformation would affect adversarial perturbation.

5.2.1 Scale Consistency Property

Scale theory is commonly applied in image segmentation task [101], and therefore we train scale resilient models to obtain robust ones, which we perform attacks against. On these scale resilient models, we first analyze how image scaling affect segmentation results for benign/adversarial samples. We applied the DAG [132] and Houdili [27] attacks against the DRN and DLA models with different adversarial targets. The images and corresponding segmentation results before and after scaling are shown in Figure 5.5. We apply Gaussian kernel with different standard deviations (std) to scale both benign and adversarial instances. It is clear that when we apply Gaussian blurring with higher std (3 and 5), adversarial perturbation is harmed and the segmentation results are not

longer adversarial targets for scale transformed adversarial examples as shown in Figure 5.5 (a)-(e).



(a) Benign example

(b) DAG — Kitty

(c) DAG — Pure

(d) Houdini — Kitty

(e) Houdini — Pure

Figure 5.5: Examples of images and corresponding segmentation results before/after image scaling on Cityscapes against DRN model. For each subfigure, the first column shows benign/adversarial image, while the later columns represent images after scaling by applying Gaussian kernel with std as 0.5, 3, and 5, respectively. (a) shows benign images before/after image scaling and the corresponding segmentation results; (b)-(e) present similar results for adversarial images generated by DAG and Houdini attacks targeting on Kitty and Pure.

5.3 Experimental Results

In this section, we conduct comprehensive large scale experiments to evaluate the image spatial and scale consistency information for benign and adversarial examples generated by different attack methods. We will also show that the spatial consistency based detection method is robust against sophisticated adversaries with knowledge about defenders, while scale transformation method is not.

5.3.1 Implementation Details

Datasets. We apply both Cityscapes [28] and BDD100K [140] in our evaluation. We show results on the validation set of both datasets, which contains 500 high resolution images with a combined 19 categories of segmentation labels. These two datasets are both outdoor datasets containing

instance-level annotations, which would raise real-wold safety concerns if they were attacked. Comparing with other datasets such as Pascal VOC [37] and CamVid [12], these two dataset are more challenging due to the relatively high resolution and diverse scenes within each image.

Semantic Segmentation Models. We apply Dilated residual networks (DRN) [138] and Deep Layer Aggregation (DLA) [139] as our target models. More specifically, we select DRN-D-22 and DLA-34. For both models, we use 512 crop size and 2 random scale during training to obtain scale resilient models for both the BDD and Cityscapes datasets. The mIOU of these two models on pristine training data are shown in Table 5.1. More result on different models can be found in appendix.

Adversarial Examples We generate adversarial examples based on two state-of-the-art attack methods: DAG [132] and Houdini [27] using our own implementation of the methods. We select a complex image, Hello Kitty (Kitty), with different background colors and a random pure color (Pure) as our targets on Cityscapes dataset. Furthermore, in order to increase the diversity, we also select a real-world driving scene (Scene) without any cars from the BDD training dataset as another malicious target on BDD. Such attacks potentially show that every image taken in the real world can be attacked to the same scene without any car showing on the road, which raises great security concerns for future autonomous driving systems. Furthermore, we also add three additional adversarial targets, including "ECCV 2018", "Remapping", and "Color strip" in appendix to increase the diversity of adversarial targets.

We generate 500 adversarial examples for Cityscapes and BDD100K datasets against both DRN and DLA segmentation models targeting on various malicious targets (More results can be found in appendix).

5.3.2 Spatial Consistency Analysis

To evaluate the spatial consistency analysis quantitatively for segmentation task, we leverage it to build up a simple detector to demonstrate its property. Here we perform patch based spatial consistency analysis, and we select patch size and region bound as $s = 512$, $b_{low} = 32, b_{upper} = 64$. We select the number of overlapping regions as $K \in \{1, 5, 10, 50\}$. Here we first select some benign instances, and calculate the normalize mIOU of overlapping regions from two random patches. We record the lower bound of theses mIOU as the threshold of the detection method. Note that when reporting detection rate in the rest of the chapter, we will use the threshold learned from a set of benign training data; while we also report Area Under Curve (AUC) of Receiver Operating Characteristic Curve (ROC) curve of a detection method to evaluate its overall performance. Therefore, given an image, for each overlapping region of two random patches, we will calculate

the normalize mIOU and compare with the threshold calculated before. If it is larger, the image is recognized as benign; vice versa. This process is illustrated in Algorithm 2. We report the detection results in terms of AUC in Table 5.1 for adversarial examples generated in various settings as mentioned above. We observed that such simple detection method based on spatial consistency information can achieve AUC as nearly 100% for adversarial examples that we studied here. In addition, we also select s with a random number between 384 to 512 (too small patch size will affect the segmentation accuracy even on benign instances, so we tend not to choose small patches on the purpose of control variable) and show the result in appendix. We observe that random patch sizes achieve similar detection result.

Method		Model	mIOU	Detection				Detection Adap			
				DAG		Houdini		DAG		Houdini	
				Pure	Kitty	Pure	Kitty	Pure	Kitty	Pure	Kitty
Scale (std)	0.5	DRN (16.4M)	66.7	100%	95%	100%	99%	100%	67%	100%	78%
	3.0			100%	100%	100%	100%	100%	0%	97%	0%
	5.0			100%	100%	100%	100%	100%	0%	71%	0%
	0.5	DLA (18.1M)	74.5	100%	98%	100%	100%	100%	75%	100%	81%
	3.0			100%	100%	100%	100%	100%	24%	100%	34%
	5.0			100%	100%	100%	100%	97%	0%	95%	0%
Spatial (K)	1	DRN (16.4M)	66.7	91%	91%	94%	92%	98%	94%	92%	94%
	5			100%	100%	100%	100%	100%	100%	100%	100%
	10			100%	100%	100%	100%	100%	100%	100%	100%
	50			100%	100%	100%	100%	100%	100%	100%	100%
	1	DLA (18.1M)	74.5	96%	98%	97%	97%	99%	99%	100%	100%
	5			100%	100%	100%	100%	100%	100%	100%	100%
	10			100%	100%	100%	100%	100%	100%	100%	100%
	50			100%	100%	100%	100%	100%	100%	100%	100%

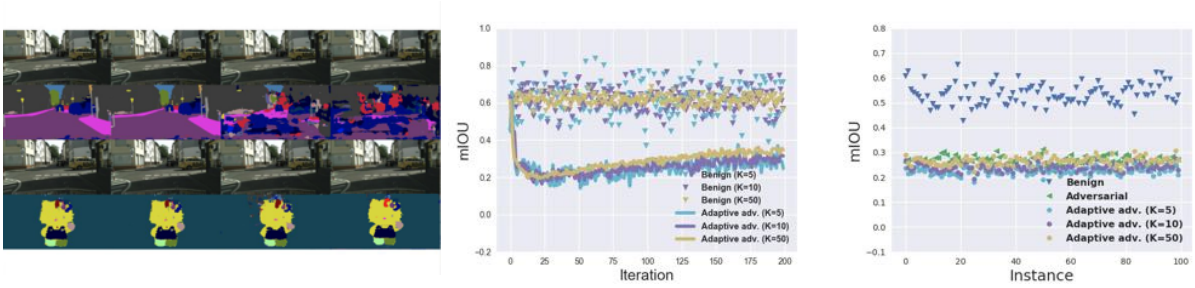
Table 5.1: Detection results (AUC) of image spatial (Spatial) and scale consistency (Scale) based methods on Cityscapes dataset. The number in parentheses of the Model shows the number of parameters for the target mode, and mIOU shows the performance of segmentation model on pristine data. We color all the AUC less than 80% with red.

5.3.3 Image Scale Analysis

As a baseline, we also utilize image scale information to perform as a simple detection method and compare it with the spatial consistency based method. We apply Gaussian kernel to perform the image scaling based detection, and select $\mathbf{std}_{detect} \in \{0.5, 3, 5\}$ as the standard deviation of Gaussian kernel. We compute the normalize mIOU between the original and scalled images. Similarly, the detection results of corresponding AUC are shown in Table 5.1. It is demonstrated

that detection method based on image scale information can achieve similarly high AUC compared with spatial consistency based method.

5.3.4 Adaptive Attack Evaluation



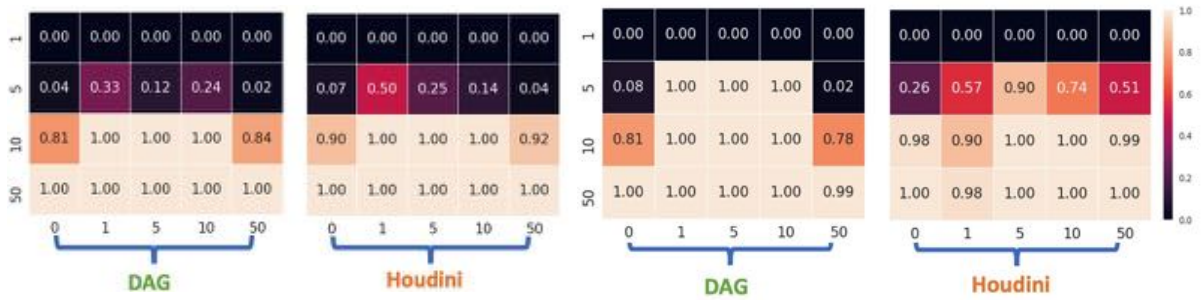
(a) Image scaling (b) Convergence analysis (c) spatial consistency

Figure 5.6: Performance of adaptive attack. (a) shows adversarial image and corresponding segmentation result for adaptive attack against image scaling. The first two rows show benign images and the corresponding segmentation results; the last two rows show the adaptive adversarial images and corresponding segmentation results under different std of Gaussian kernel (0.5, 3, 5 for column 2-4). (b) and (c) show the performance of adaptive attack against spatial consistency based method with different K. (b) presents mIOU of overlapping regions for benign and adversarial images during along different iterations. (c) shows mIOU for overlapping regions of benign and adversarial instances at iteration 200.

Regarding the above detection analysis, it is important to evaluate *adaptive attacks*, where adversaries have knowledge of the detection strategy.

As Carlini & Wagner suggest [17], we conduct attacks with full access to the detection model to evaluate the adaptive adversary based on Kerckhoffs principle [103]. To perform adaptive attack against the image scaling detection mechanism, instead of attacking the original model, we add another convolutional layer after the input layer of the target model similarly with [17]. We select std $\in \{0.5, 3, 5\}$ to apply adaptive attack, which is the same with the detection model. To guarantee that the attack methods will converge, when performing the adaptive attacks, we select 0.06 for the upper bound for adversarial perturbation, in terms of L_2 distance (pixel values are in range [0,1]), since larger than that the perturbation is already very visible. The detection results against such adaptive attacks are shown in Table 5.1 on Cityscapes (We omit the results on BDD to appendix). Results on adaptive attack show that the image scale based detection method is easily to be attacked (AUC of detection drops dramatically), which draws similar conclusions as in classification task [17]. We show the qualitative results in Figure 5.6 (a), and it is obvious that even under large std of Gaussian kernel, the adversarial example can still be fooled into the malicious target (Kitty).

Next, we will apply adaptive attack against the spatial consistency based method. Due to the



(a) Kitty

(b) Pure

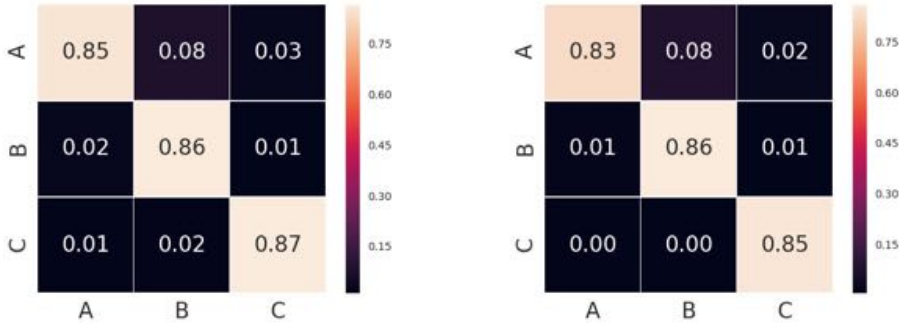
Figure 5.7: Detection performance of spatial consistency based method against adaptive attack with different K on Cityscapes with DRN model. X-axis indicates the number of patches selected to perform the adaptive attack (0 means regular attack). Y-axis indicates the number of overlapping regions selected for during detection.

randomness of the approach, we propose to develop a strong adaptive adversary that we can think of by randomly select K patches (the same value of K used by defender). Then the adversary will try to attack both the whole image and the selected K patches to the corresponding part of malicious target. The detailed attack algorithm is shown in the supplementry materials. The corresponding detection results of the spatial consistency based method against such adaptive attacks on Cityscapes are shown in Table 5.1. It is interesting to see that even against such strong adaptive attacks, the spatial consistency based method can still achieve nearly 100% detection results. We hypothesize that it is because of the high dimension randomness induced by the spatial consistency based method since the search space for patches and the overlapping regions is pretty high. Figure 5.6 (b) analyzes the convergence of such adaptive attack against spatial consistency based method. From figure 5.6 (b) and (c), we can see that with different K, the selected overlapping regions still remain inconsistent with high probability.

Since the spatial consistency based method can induce large randomness, we generate a confusion matrix of detection results for adversaries and detection method choosing various K as shown in Figure 5.7. It is clear that for different malicious targets and attack methods, choosing $K = 50$ is already sufficient to detect sophisticated attacks. In addition, based on our empirical observation, attacking with higher K increases the computation complexity of adversaries dramatically.

5.3.5 Transferability Analysis

Given the common properties of adversarial examples for both classifier and segmentation tasks, next we will analyze whether transferability of adversarial examples exists in segmentation models considering they are particularly sensitive to spatial and scale information. *Transferability* is demonstrated to be one of the most interesting properties of adversarial examples in classification



(a) DAG (b) Houdini

Figure 5.8: Transferability analysis: cell (i, j) shows the normalized mIoU value or pixel-wise attack success rate of adversarial examples generated against model j and evaluate on model i. Model A,B,C are DRN (DRN-D-22) with different initialization. We select "Hello Kitty" as target

task, where adversarial examples generated against one model is able to mislead the other model, even if the two models are of different architectures. Given this property, transferability has become the foundation of a lot of black-box attacks in classification task. Here we aim to analyze whether adversarial examples in segmentation task still retain high transferability. First, we train three DRN models with the same architecture (DRN-D-22) but different initialization and generate adversarial images with the same target.

Each adversarial image has at least 96% pixel-wise attack success rate against the original model. We evaluate both the DAG and Houdini attacks and evaluate the transferability using normalized mIoU excluding pixels with the same label for the ground truth adversarial target. We show the transferability evaluation among different models in the confusion matrices in Figure 5.8[1]. We observe that the transferability rarely appears in the segmentation task. More results on different network architectures and data sets are in the appendix.

As comparison with classification task, for each network architecture we train a classifier on it and evaluate the transferability results as shown in appendix. As a control experiments, we observe that classifiers with the same architecture still have high transferability aligned with existing findings, which shows that the low transferability is indeed due to the natural of segmentation instead of certain network architectures.

This observation here is quite interesting, which indicates that black-box attacks against segmentation models may be more challenging. Furthermore, the reason for such low transferability in segmentation is possibly because adversarial perturbation added to one image could have focused on a certain region, while such spatial context information is captured differently among different

[1]Since the prediction of certain classes presents low IoU value due to imperfect segmentation, we eliminate K classes with the lowest IoU values to avoid side effects. In our experiments, we set K to be 13.

models. We plan to analyze the actual reason for low transferability in segmentation in the future work.

5.4 Conclusions

Adversarial examples have been heavily studied recently, pointing out vulnerabilities of deep neural networks and raising a lot of security concerns. However, most of such studies are focusing on image classification problems, and in this chapter we aim to explore the spatial context information used in semantic segmentation task to better understand adversarial examples in segmentation scenarios. We propose to apply spatial consistency information analysis to recognize adversarial examples in segmentation, which has not been considered in either image classification or segmentation as a potential detection mechanism. We show that such spatial consistency information is different for adversarial and benign instances and can be potentially leveraged to detect adversarial examples even when facing strong adaptive attackers. These observations open a wide door for future research to explore diverse properties of adversarial examples under various scenarios and develop new attacks to understand the vulnerabilities of DNNs.

CHAPTER 6

Detecting Adversarial Examples by Using the Property of the Data

In this chapter, we show how to leverage properties of the data type to detect adversarial examples.

While currently most research on adversarial examples focuses on static images, DNNs on videos is a particularly interesting and important domain, as attacks against many applications have the potential to cause serious physical and financial damage. For example, one application of DNNs video is in autonomous vehicles; DNNs are used to identify other cars, road markings, street signs, and pedestrians. An adversarial attack forcing a network to classify a stop sign as a speed limit sign could easily cause a crash. Recently Wei et.al [120] proposed an adversarial attack targeting on action recognition task in videos which again emphasizes the vulnerabilities of learners for videos.

Several defense or detection methods have been proposed on static images but most of them are defeated by adaptive attacks [6, 17, 84, 85]. As a result, directly applying existing defenses on static images to videos is not robust nor efficient considering the high requirements for video processing. While the general defense approach is hard, leveraging special properties of data source (e.g. videos) and enhancing model robustness is possible.

In this chapter, we introduce *AdvIT* : the adversarial frame identifier for videos by leveraging the temporal consistency of the video. In particular, we allow the attacker to have white-box access to the target DNNs and add adversarial perturbation to one or more frames. Here we consider two types of attacks: *independent frame attack* which adds adversarial perturbation to selected frames independently (e.g. Houdini and DAG [27, 132]); and *temporal continuity attack* which generates perturbation considering the continuity among video frames (e.g. sparse and universal attack [120]). Our temporal consistency-based detection framework *AdvIT* is shown in Fig. 6.1. Given a target frame within a video, we first estimate the optical flows between the target x_t and its previous frames $(x_{t-1}, \cdots, x_{t-k})$. We then fuzz the estimated optical flows with small randomness $\alpha \sim \mathcal{N}(0, \sigma^2)$ and transform the previous frames as "pseudo frames" and check the consistency between the outputs of learning tasks based on the pseudo frames and the original target. We find that the frame transformation described above preserves the temporal consistency of learning results if the target frame is benign, while weakens the pseudo frames' adversarial behaviour if the target

frame is adversarial. This process is independent with the fact that whether the previous frames are adversarial or not. Therefore, we can leverage the prediction results of the pseudo frame as a reference and check its consistency to detect if the target frame is adversarial.

To demonstrate the effectiveness of *AdvIT* , we test our approach on four major video based tasks including semantics segmentation, human pose estimation, object detection and action recognition. Different state of the art attack approaches including Houdini, DAG, Sparse adversarial perturbation [27, 120, 132] are evaluated. We show that our approach can detect adversarial frames with above 95% detection rate. We also show that because of the large sample space of randomness added to the optical flow, even attacks that are aware of our detector would be unfeasible. As shown in the experimental section, our detection pipeline remains robust even under the proposed strategic adaptive attacks. We perform transferability analysis for different optical flow estimators and demonstrate that it is hard to transfer perturbation generated against one estimator to another. In addition, we analyze the performance of the optical flow estimator and conclude that our approach does not require an accurate estimator to achieve a high detection rate.

Our detection approach has several advantages compared to existing approaches: 1) we do not require time-consuming retraining of machine learning models as most of the existing defenses [85, 112]; 2) our detector does not compromise the performance of the learning tasks; 3) due to the randomness injected, it is hard to perform adaptive attacks against the detection method; 4) we do not require the optical flow estimation to be differentiable, which ensures its wide application;

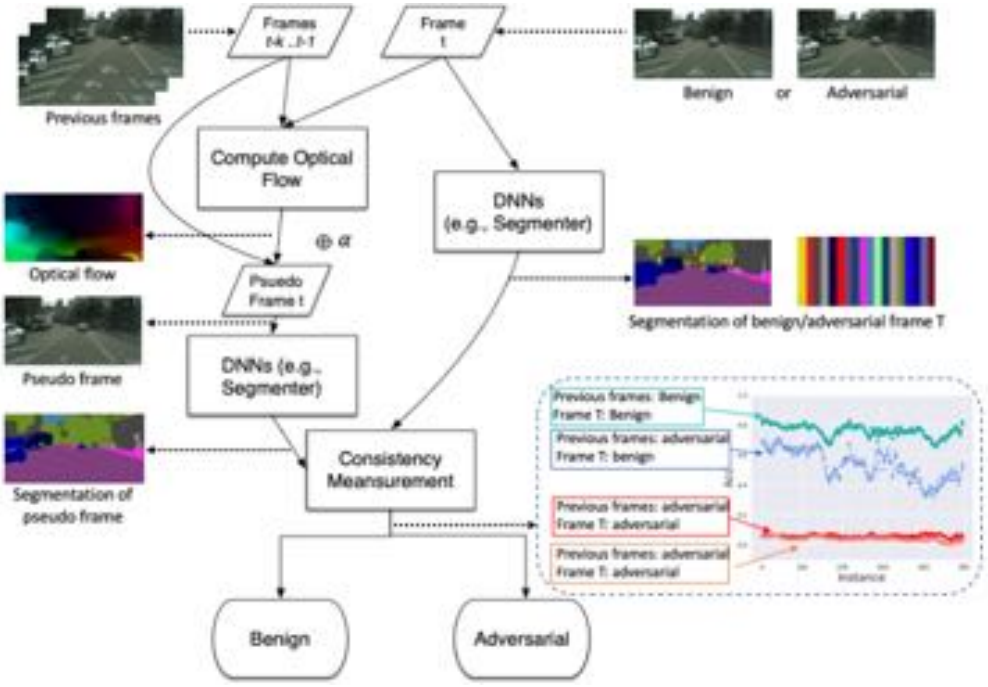


Figure 6.1: Pipeline of the proposed temporal consistency based adversarial frame identifier: *AdvIT* .

6.1 Adversarial frame identifier via temporal consistency: *AdvIT*

In this section, we first formally define the problem. Then, we provide an overview of the proposed approach advIT, and discuss each step in detail.

Formally, we define the problem as follows: Let $x_1, \ldots, x_t$ be the sequence of image frames of a (streaming) video, and x_t is the target frame. Let g be a learner with output $g(x_t) = y_t$. In this context, the attacker can inject small perturbation (*i.e.*, $x_i \leftarrow x_i + \boldsymbol{\epsilon}_i$) to one or more frames to achieve $g(x_t) = y^*$ where y^* is the adversarial target depending on the learning task. Our goal is to determine whether the target frame x_t is adversarial without any other knowledge except for the previous k frames $x_{t-k}, \ldots, x_{t-1}$, and it is not clear whether the previous k frame are benign or adversarial.

Threat Model In this work, we mainly focus on two types of frame based adversarial attacks: *independent frame attack* and *temporal continuity attack*, both of which aim to add perturbation to one or more frames and therefore mislead the target learner. We believe such frame based attack could lead to severe consequences given the fact that frame based approaches are the most effective and commonly used ones in videos [14, 98].

In particular, *independent frame attack* includes Houdini [27] and DAG [132] attacks for video segmentation, object detection and human pose estimation tasks; and *temporal continuity attack* includes Sparse attack [120] on action recognition and Universal perturbation [92] on the previous three tasks.

To avoid trivial detection, an adversary needs to constraint the magnitude of the perturbation. Our goal is to detect adversarial examples generated by previous attacks, which is $\mathcal{L}_p$-based attacks. Without loss of generality, we use l_2 to bound the added perturbation. Some video attack examples for the considered four learning tasks are shown in Fig 6.2 and Fig 6.3.

Figure 6.2: Benign and adversarial frames generated on Cityscapes and Davis Challenge 17 dataset for video segmentation and human pose estimation respectively.

Figure 6.3: Benign and adversarial frames generated on MPII and UCF-101 for video object detection and action recognition respectively.

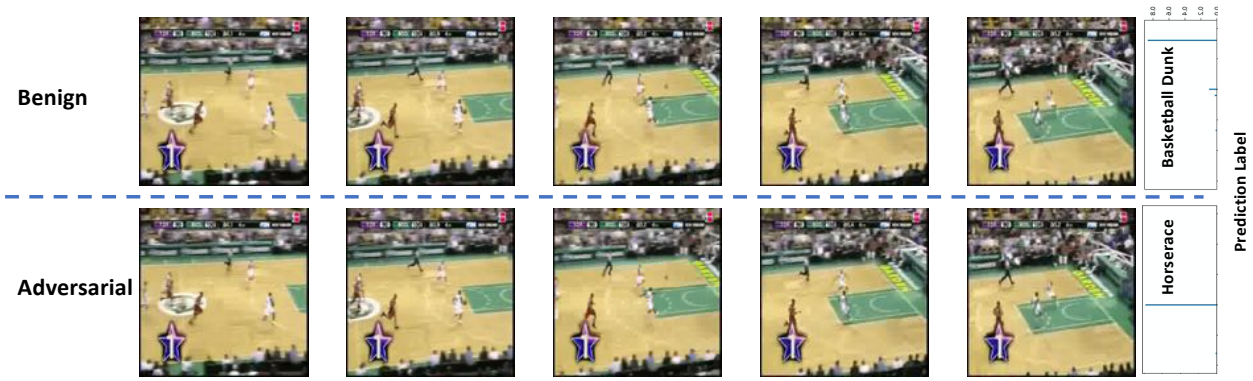


Figure 6.4: Benign and adversarial frames generated on MPII and UCF-101 for action recognition.

6.1.1 Overview of Method

Our detection algorithm is based on temporal consistency of videos. Since the next frame in the video is the continuation of the objects and the scene after their small movements, they can be mostly reconstructed from the current frame if we can compute such movements. Due to this continuity, we should have consistent learning outputs from neighbor frames [55]. However, this may not be true when the next frame contains adversarial perturbation which may interrupt such property.

We call the reconstructed frame as a *"pseudo frame"*, and can *validate* the learning output temporal consistency by comparing the output of the *to-be-verified* target frame and the pseudo frame. Specifically, we observe the following properties regarding the validation of the temporal consistency with the pseudo frame. First, since adversarial perturbation is very specific to the frame, a newly reconstructed pseudo frame is much less affected by the adversarial perturbation. Its output is very close to that of a benign frame. Second, adversarial perturbation in the target frame breaks

the output temporal continuity compared to the output of the pseudo frame. Thus, if the target frame is adversarial, we can observe that the temporal consistency of the output does not hold.

Based on this observation, we propose the adversarial frame detection framework that tests temporal consistency of the outputs as shown in Fig 6.1. First, we generate pseudo frames based on each of the k previous frames ($x_{t-k} \cdots x_{t-1}$) by estimating optical flow (O_F) from each to the target frame X_t, and adding certain random transformation $\alpha \sim \mathcal{N}(0, \sigma^2)$. Then, we run the learner (e.g. segmenter) on the pseudo frames and the target frame to get prediction results. Finally, we compare their results to test if the temporal consistency is satisfied. Note that our method is independent of the adversarial behaviour of the previous frames: they can either be adversarial or benign. The bottom-right box in Fig. 6.1 shows the four scenarios. Next, we will introduce the two components of the proposed method in detail: (1) Pseudo frame generation; (2) temporal consistency based test.

Pseudo Frame Generation We combine two types of transformations to generate the pseudo frames: optical flow and random transformation. First, optical flow is a transformation caused by movements of the viewer or the objects in the scene. This technique is able to reconstruct subsequent frames and is the basis of a number of video codecs, such as MPEG-2 [47]. Machine learning models have also successfully been trained to generate the vector field $\boldsymbol{V}$ when trained on videos [33, 55]. In particular, optical flow has been applied to estimate the instantaneous 2D velocity of visible surface points from time-varying image signal. Traditional optical flow estimation methods usually involve solving optimization problems based on assumptions of consistency, smoothness of intensity, and gradients [78].

An optical flow estimator is a function F which generates a vector field V indicating the direction and distance to move pixels within an image. Deep-learning-based optical flow estimation models have been widely studied on different video tasks. Dosovitskiy et.al [33] firstly applied deep neural network to estimate the optical flow. Here we leverage the DNNs based optical flow [55] to characterize the continuity among video frames. Given two temporally close frames, we can quantify the motion using an optical flow estimation algorithm. By applying the optical flow to the first input frame, we can reconstruct the second one.

Formally, let x_s and x_t be a pair of temporally close frames in a video. An optical flow between the two frames is a vector field $O_F = (\boldsymbol{\Delta u}, \boldsymbol{\Delta v})$ that describes the displacement of pixels between the frames and we denote an image generated by applying flow as $\hat{x}_{s \to t}$. The goal of an optical flow algorithm is to minimize the error of the generated image $\hat{x}_{s \to t}$ and the actual frame x_t. We obtain $\hat{x}_{s \to t}$ by sampling pixel intensities from x_s; the pixel in $\hat{x}_{s \to t}$ at location (i, j) corresponds to the pixel at location $(u, v) = (i + \boldsymbol{\Delta u}(i, j), j + \boldsymbol{\Delta v}(i, j))$ in image x_s. As $(\boldsymbol{\Delta u}, \boldsymbol{\Delta v})$ can be fractional numbers and (u, v) does not necessarily lie on the integer coordinate grid, the pixel intensity can be

sampled via bilinear sampling.

$$\hat{x}_{s\to t}(i,j) = \sum_{(i',j')\in N(u,v)} x_s(i',j')(1-|u-i'|)(1-|v-j'|) \tag{6.1}$$

where $N(u,v)$ stands for the indices of the 4-pixel neighbors at location (u,v) (top-left, top-right, bottom-left, bottom-right). $x.(i,j)$ represents the pixel value at location (i,j).

To further combat the creation of adversarial perturbation, we add randomness $\boldsymbol{\alpha} \sim \mathcal{N}(0,\sigma^2)$ to the flow field $(\boldsymbol{\Delta u}, \boldsymbol{\Delta v})$ to generate the pseudo frames. This randomness can also help make adaptive attacks harder, where an adversary has full knowledge about the detection.

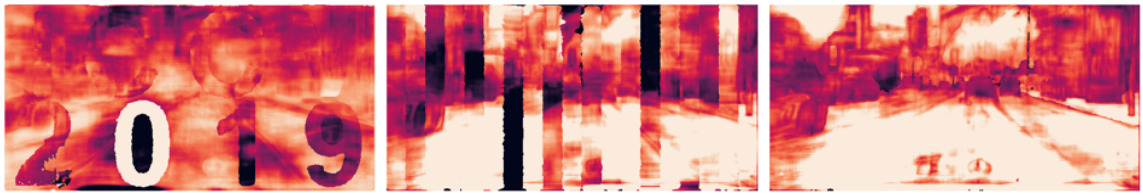
Algorithm 3: Temporal Consistency Based Test

input: target frame in a videox_t;
previous K frames of x_t: $x_{t-k}, \ldots, x_{t-1}$;
optical flow estimation model **flow**;
machine learning model g;
consistency evaluation function f;
output: Continuity metric c;

Initialization : **cs** $\leftarrow$[], $w \leftarrow x.width, h \leftarrow x.height, y_t \leftarrow g(x_t)$;
for $s \leftarrow t-1$ **to** $t-k$ **do**
  $(\boldsymbol{\Delta u}, \boldsymbol{\Delta v}) \leftarrow \mathbf{flow}(x_s, x_t)$;
  /* add randomness to optimal flow */;
  $(\boldsymbol{\Delta \tilde{u}}, \boldsymbol{\Delta \tilde{v}}) \leftarrow (\boldsymbol{\Delta u}, \boldsymbol{\Delta v}) + \boldsymbol{\alpha}$;
  /* generate pseudo frame x_{T-k} */;
  $\hat{\boldsymbol{X}}_{s\to t} \leftarrow \mathbf{warp}((\boldsymbol{\Delta \tilde{u}}, \boldsymbol{\Delta \tilde{v}}), x_s)$;
  $\hat{\boldsymbol{Y}}_{s\to t} \leftarrow g(\hat{\boldsymbol{X}}_{s\to t})$;
  /* measure consistency information */;
  cs $\overset{+}{\leftarrow} f(\hat{\boldsymbol{Y}}_{s\to t}, y_t)$;
end
$c \leftarrow \mathbf{Mean}(\mathbf{cs})$;
Return: c

Temporal Consistency based Test To quantitatively demonstrate the difference of temporal consistency between adversarial and benign cases, we first use semantic segmentation as an example here to illustrate our findings. Suppose given $k+1$ consecutive frames from video, $x_{t-k}, \ldots, x_{t-1}, x_t$, we estimate the O_F between each of the previous k frames x_s and x_t, where $s = t-k, ..., t-1$. Next, we add randomness α to each O_F to reconstruct pseudo frame $\hat{\boldsymbol{X}}_{s\to t}$ from x_s. These pseudo frames and x_t are then sent as input to learning model g. We normalize the output of g by the softmax function so that the prediction of every pixel is a vector indicating the probability of the pixel belonging to every class. The prediction results of the learning models are denoted by $\hat{\boldsymbol{Y}}^{\boldsymbol{d}}_{s\to t}$

(a) Benign frame (b) Heatmap of a benign frame



(c) ICCV (d) Stripe (e) Remapping

Figure 6.5: Heatmap of per-pixel cross-entropy. (a) and (b) show a benign frame and the corresponding per-pixel cross entropy between the prediction of its pseudo frames and itself. The rest show similar per-pixel cross entropy for adversarial frames with different targets. The labels indicate their adversarial targets.

and $\boldsymbol{Y}_t^{\boldsymbol{d}}$ respectively. We compute the average for the prediction vectors of k pseudo frames, $\boldsymbol{S} = \frac{1}{k}\sum_{s=t-k}^{t-1} \hat{\boldsymbol{Y}}_{s\to t}^{\boldsymbol{d}}$, and we use $\boldsymbol{S}(i,j)[m]$ to indicate the averaged probability of pixel (i,j) being predicted to be class m in pseudo frames. Based on the k pseudo frames, we calculate the cross entropy $\boldsymbol{E}$ between $\boldsymbol{S}$ and y_t as

$$\boldsymbol{E} = \sum_m -y_t^d[m] \circ \log \boldsymbol{S}[m]$$

where $\circ$ denotes Hadamard product. We visualize the cross entropy $\boldsymbol{E}$ in Fig. 6.5: Fig. 6.5c to Fig. 6.5e show the heatmaps of cross entropy when $\boldsymbol{X_t}$ is adversarial examples with different attack targets, while Fig. 6.5b shows the heatmap of cross entropy when $\boldsymbol{X_t}$ is benign. It is clear that for the benign instance, the prediction results for most pixels are consistent except for small regions around the boundaries of the objects; while for the adversarial targets, most of the pixels show inconsistent prediction results. We also observe that whether $x_{t-k}, \ldots, x_{t-1}$ are benign or adversarial has little impact on the prediction consistency for $\boldsymbol{X_t}$.

AdvIT Based on such observation, we proposed leveraging the temporal consistency information to distinguish adversarial frames in videos and provide the following detailed algorithm. Without loss of generality, we assume x_t is a current frame and our goal is to detect whether the current frame x_t is adversarial. Given an optical flow model, we denote the optical flow between previous frame x_s and x_t by $O_F = (\boldsymbol{\Delta u}, \boldsymbol{\Delta v})_{s\to t}$. Randomness $\boldsymbol{\alpha}$ is added to the optical flow $(\boldsymbol{\Delta u}, \boldsymbol{\Delta v})_{s\to t}$

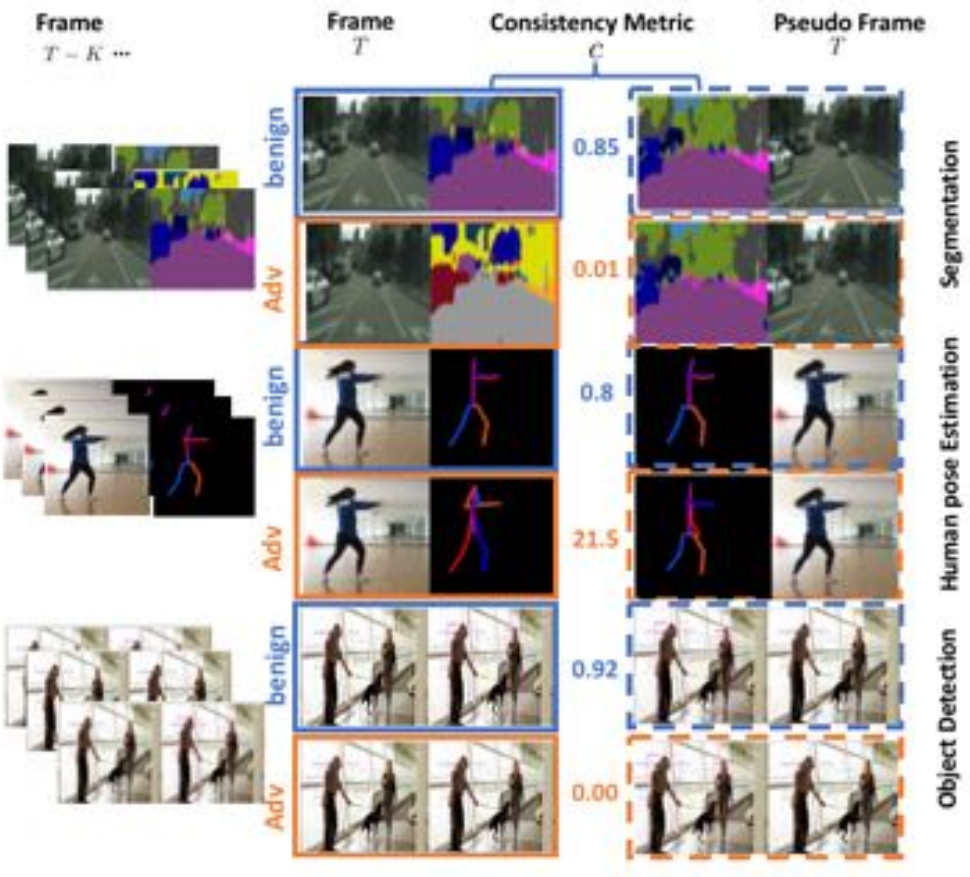


Figure 6.6: Examples of consistency measurement based on *AdvIT* for various video tasks. The first column indicates previous Frames. The second column indicates the current frame and corresponding prediction result. The last column indicates a sampled pseudo frame and corresponding prediction. The consistency metric C shows quantitative results for different learning tasks. Note that higher C for segmentation and object detection means higher consistency, while lower C indicates more consistent for Human pose estimation since it is based on L_2 distance.

to get new optical flow $(\boldsymbol{\Delta}\tilde{\boldsymbol{u}}, \boldsymbol{\Delta}\tilde{\boldsymbol{v}})_{s\to t}$ where $(\boldsymbol{\Delta}\tilde{\boldsymbol{u}}, \boldsymbol{\Delta}\tilde{\boldsymbol{v}})_{s\to t} = (\boldsymbol{u}, \boldsymbol{v})_{s\to t} + \boldsymbol{\alpha}_{\boldsymbol{u},\boldsymbol{v}}$. After obtaining $(\boldsymbol{\Delta}\tilde{\boldsymbol{u}}, \boldsymbol{\Delta}\tilde{\boldsymbol{v}})_{s\to t}$ for $s = t-k, ..., t-1$, we generate the pseudo frames $\hat{\boldsymbol{X}}_{s\to t}$. We then calculate the consistency metric c between $y_t = g(x_t)$ and $y_{s\to t} = g(\hat{\boldsymbol{X}}_{s\to t})$ respectively with a scalar consistency function f to determine whether x_t is an adversarial frame or not, where $c = \frac{1}{k}\sum_{s=t-k}^{t-1} f(y_{s\to t}, y_t)$. The algorithm of this temporal continuity based method is shown in Algorithm 3, where $\mathbf{warp}$ is achieved via bilinear sampling as defined in Eq. 6.1.

We adopt different consistency measurement functions f for various learning tasks: (1) Segmentation: Pixel-wise accuracy[1]. (2) Human Pose Estimation: Average L_2 distance over all key joints. Note that, high distance indicates low consistency. (3) Object Detection: mIoU between bounding boxes of pseudo frames and the current frame. (4) Action Recognition: the average of forward

[1]Pixel-wise accuracy and mIoU provide similar results and the former is much more computationally efficient.

and backward KL divergence between the two categorical distributions. The detailed algorithm to calculate the mIoU is shown in appendix.

Fig. 6.6 shows the examples of our detection method. We can observe that the inconsistency between the target frame and corresponding pseudo frames is high for an adversarial frame and low for benign, and this conclusion holds for various learning tasks on video.

6.2 Experimental Results

In this section, we present experimental results on detecting adversarial frames with *AdvIT* against different attacks for four learning tasks on videos, including video semantic segmentation, human pose estimation, object detection, and action recognition. We show that our adversarial frame detection method is robust even under a strong adaptive attack where the adversary has perfect knowledge of the learning model and detection mechanism.

6.2.1 Implementation Details

Semantic segmentation For the semantic segmentation task, we use CityScapes dataset which consists of high-resolution (1024x2048) outdoor videos captured from a moving car. Attacks against the CityScapes dataset pose a realistic threat to recognition models in real-world applications, in particular autonomous driving. We adopt the state-of-the-art Dilated Residual Network [138] model with DRN-D-22 architecture which is trained on the CityScapes dataset. The mean Intersection Over Union (mIoU) of this model on pristine data is 66.7. We demonstrate our results on video clips from the same dataset, each consisting of 100 high-resolution frames shot at a frame rate of 17Hz. We evaluate over three different adversarial targets: "Remapping", "Stripe", and "ICCV 2019". The details of these targets are shown in appendix. We generate adversarial frames based on two state-of-the-art attack methods: Houdini [27] and DAG [132]. Each attack was run with a maximum perturbation of $l_2 = 0.03$ with input frames scaled from $[0, 1]$ until 98% pixel-wise accuracy of the target was achieved. We select $\sigma = 0.002$ for detection.

Human Pose Estimation In the human pose estimation task, we attack the Stacked Hourglass Network model [94] trained on MPII human pose dataset [4]. The two-stack model we use is pretrained on the MPII dataset [4] and achieved a mean PCHk score of 86.95 on the MPII validation dataset at the threshold of 0.5. We attacked 3 clips of video data from the MPII human pose estimation dataset and YouTube [2] using the Houdini algorithm which is the current the-state-of-art. The attack targets we choose are "Transpose" and "Shuffle". "Transpose" means transposing coordinates of benign image predictions. "Shuffle" means shuffling the joints of the pose predictions. We select $\sigma = 0.02$ during detection.

Object Detection We use YOLOv3 [98] as our target model for the object detection task. We select two video clips randomly from DAVIS Challenge 17 dataset [1] to perform attacks on. We select two targets for simplicity: "All" and "Person". "All" means removing all of the bounding boxes in the images while "Person" means removing only the bounding boxes of persons in images. Such attacks potentially show that every image taken in the real surveillance system can be attacked to the scene without any object or without any person which brings severe security concerns for current surveillance systems based on object detection algorithms. We use DAG algorithm which is the current the-state-of-the-art algorithm, to attack the YOLOv3. We run the attack with a maximum perturbation of 0.03 with input frames scaled from $[0, 1]$ until we achieve 100% removal of the target objects. We select $\sigma = 0.002$ during detection.

Action Recognition This task and the corresponding attack method Sparse takes the video temporal continuity into account. The target model makes action predictions based on a whole clip of a video instead of processing individual frame, i.e.

$$Y = F(X_1, X_2, \ldots, X_N)$$

We use the CNN+RNN model used in [120] as video action recognition model and also apply the state of the art attack Sparse to generate adversarial video clips. The model is trained on the UCF-101 dataset [105] to predict the action from 101 classes, and uses a Inception V2 model [109] to extract features from each frame. Given a recognition model $\mathbf{F}_\theta$, several of video clips $\{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_N\}$ and their corresponding adversarial target labels $\{y_1, y_2, \ldots, y_N\}$, the attack optimizes the following objective:

$$\underset{\mathbf{E}}{\operatorname{argmin}}\, \lambda||\mathbf{M} \cdot \mathbf{E}||_p + \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{1}_{y_i^*}, \mathbf{F}_\theta(\mathbf{C}_i + \mathbf{M} \cdot \mathbf{E}))$$

where $\mathbf{E}$ is a universal adversarial perturbation and $\mathbf{M}$ is a predefined temporal mask to enforce sparsity. $l(\cdot)$ represents the classification loss. The attack aims to generate a universal perturbation for all N clips, and in our experiment the recognition model mis-classified 13245 videos out of 13320 samples. We use *AdvIT* to detect adversarial video clips instead of adversarial frames for this task.

6.2.2 Temporal Consistency Based Detection

Here we evaluate the detection performance of *AdvIT* on different video tasks comparing with other baseline methods. Following Algorithm 3, given a frame, we first generate pseudo frames for its previous k frames and then calculate the consistency metric between the frame and these pseudo frames. Based on the distribution of consistency metric C, we identify the adversarial frames if C is low, and vice versa.

Task	Attack Method	Target	Defense Method	Detection (k) 1	3	5
Semantic Segmentation	Houdini	Stripe	*Replacement*	50%	50%	50%
			JPEG	**100%**	-	-
			AdvIT	**100%**	**100%**	**100%**
Human Pose Estimation	Houdini	Shuffle	*Replacement*	50%	50%	50%
			JPEG	98%	-	-
			AdvIT	**100%**	**100%**	**100%**
Object Detection	DAG	Person	*Replacement*	50%	50%	50%
			JPEG	60%	-	-
			AdvIT	**98%**	**99%**	**100%**

Table 6.1: Comparison of detection results (AUC) against different attacks for *AdvIT* and baseline methods.

We evaluate *AdvIT* against two types of frame based adversarial attacks: *independent frame attack* and *temporal continuity attack* respectively. For both scenarios, we report the Area Under Curve (AUC) of Receiver Operation Characteristic Curve (ROC) of *AdvIT* and baselines.

Detecting independent frame attack *Independent frame attack* includes Houdini [27] and DAG [132] on three video tasks: semantic segmentation, human pose estimate and object detection. Since each frame is independent for attackers, the attacker can decide whether to attack the previous frame so the previous frames can be either adversarial or benign. Our method aims to identify whether the current frame is adversarial, regardless of the status of previous frames. Therefore, we test our method and report the results under various conditions: previous frames are purely benign, adversarial, or mixture. The result is shown in Tab. 6.1 and other result is shown in appendix. Note that as this is the first work to detect adversarial frames within a video, there is no existing detection methods dedicated to video to compare with. To demonstrate the effectiveness of *AdvIT* , we, instead, compared our method with two baselines: a traditional static image based method, JPEG compression [30, 36, 45] shown as *JPEG* and *Replacement* in Tab. 6.1. *JPEG* compresses the current frame to generate a "pseudo frame" and then calculates the consistency metric between the current and "pseudo frame". *Replacement* directly leverages the temporal continuity by replacing the current frame with previous frames to generate "pseudo frames". It then calculates the continuity metrics between the current frame and the "pseudo frame". We evaluate the performance by aggregating previous k frames, where $k \in \{1, 3, 5\}$. Note that *JPEG* only considers the current frame so it only applies for $k = 1$.

From Tab. 6.1. We also observed that even *JPEG* achieves a high detection rate on Semantic segmentation and Human pose estimation, it is less robust than *AdvIT* and performs worse on object detection. It indicated that the perturbation on object detection might be subtle against compression. Compared with the baseline methods, *AdvIT* shows promising detection performance (almost 100%) against independent frame attacks on various learning tasks under different scenarios. (More

detection results against different attack targets are omitted in appendix.)

To further illustrate the effectiveness of *AdvIT* , we show *AdvIT* on the two extreme cases: previous frames are adversarial or benign. We observe that *AdvIT* achieve almost 100% success rate for identifying adversarial frames all kinds of settings without requiring knowledge about whether previous frames are adversarial. The complete results are deferred to the appendix.

Detecting temporal continuity attack Considering attacks that also take temporal continuity of videos into account, we evaluate the effectiveness of *AdvIT* on *temporal continuity attack*, Sparse attack [120] on video action recognition and universal perturbation for the previous three tasks. For universal perturbation, we generate universal perturbation for 5 frames within videos. We extend the DAG and Houdini methods to generate universal perturbation. Tab. 6.2 shows the detection results of *AdvIT* against such attacks. We observe that *AdvIT* can defend against the universal perturbation with 100% detection rate on different video tasks, which implies that universal perturbation can not transfer to pseudo frames. Though Sparse attack [120] aims to generate sparse and continuous perturbation for videos, *AdvIT* can still achieve a high detection rate by leveraging both temporal consistency and randomness. Note that the detection rate increases slightly with the growth of k, but there is no need to carefully tune k since $k = 1$ already achieves detection rate above 95%. In addition, to analyze the effectiveness of *AdvIT* against the adversarial attack with different strengths, we conduct an experiment by limiting the perturbation magnitude to 2, 16, 32 pixels (in range of [0,255]). The detailed results are shown in appendix. It shows that the detection rate will decrease a bit with the magnitude increasing. But with ensemble of previous k frames, it is still effective.

Task	Attack Method	Target	Detection (k)		
			1	3	5
Semantic Segmentation	Universal	Strip	100%	100%	100%
Human Pose Estimation		shuffle	100%	100%	100%
Object Detection		all	100%	100%	100%
Action Recognition	Sparse	-	95%	96%	97%

Table 6.2: Detection results (AUC) against *temporal continuity attack*

6.2.3 Analysis of Adaptive Attacks

Given a detection method, it is important to evaluate it against a strong adaptive attacker who is aware of the detection mechanism. Thus, we conduct experiments to simulate the strong adaptive attacker we can think of to evaluate the robustness of *AdvIT* , assuming the attacker has complete access to our fully differentiable models. First, the attacker generates a perturbation that considers both the current and generated pseudo frames. Implementation details of the adaptive attack will

be included in the appendix. Such attack will fail since during detection as we add randomness α to make the optical flow estimation harder. Thus, we allow the attacker to use the state of the art adaptive attack estimation method *expectation of transformation* to approximate potential randomness [6]. We follow the setting in [6], and randomly select 30 possible α in each iteration to optimize the perturbation. We select $l_2 = 0.03$ as the upper bound of the adversarial perturbation (pixel values are in range [0,1]), as perturbation larger than that would produce noticeable visual changes to human. The detection results against such adaptive attacks among different video tasks are shown in Tab. 6.3 as "Detection Adap. We observe that *AdvIT* can still achieve above 95% detection rate. We hypothesize that it is because (a) the high dimension of spatial randomness introduces large search space; (b) indirect changes to the pseudo frames during attack are not sufficient to manipulate the prediction of both the pseudo frames and target one.

Task	Target	Previous Frames	Detection Adap (k)			Detection Trans (k) (non-differential flow)		
			1	3	5	1	3	5
Semantic Segmentation	ICCV	Benign	100%	100%	100%	100%	100%	100%
		Adversarial	95%	97%	100%	100%	100%	100%
	Remapping	Benign	100%	100%	100 %	100%	100%	100%
		Adversarial	96%	96%	98%	100 %	100%	100%
Human pose estimation	Shuffle	Benign	96%	97%	97%	100%	100%	100%
		Adversarial	94%	97%	100%	100%	100%	100%
	Transpose	Benign	98%	99%	100 %	100%	100%	100%
		Adversarial	95%	95%	100 %	100 %	100%	100%
object detection	All	Benign	99%	100%	100%	100%	100%	100%
		Adversarial	99%	100%	100%	100%	100%	100%
	Person	Benign	98%	99%	100 %	100%	100%	100%
		Adversarial	95%	96%	97%	100 %	100%	100%

Table 6.3: Detection results (AUC) of adaptive attacks and transferability analysis.

Transferbility analysis In addition to the randomness for optical flow estimation process, in this section we try to analyze the transferability of the perturbation between different flow estimator. For instance, we allow the defender to use a non-differentiable flow estimator, while the attacker uses a differentiable one for the sake of attack convenience.

We substitute a non-differential flow estimator proposed by [78] in Algorithm 3 and evaluate its performance against an adversary who can approximate flow using a differential flow estimator FlowNet [33]. Such transferability based detection results are shown in Tab. 6.3 "Detection Trans (non-differentiable flow)". We can see that the transferability based adversarial perturbation generated for differential flow estimator does not transfer to non-differential flow estimator and the detection results are 100% across all k.

Optical Flow Estimator Next we will evaluate the impact of optical flow estimator on *AdvIT* . We calculate the accuracy of the applied flow estimator for different learning tasks, and the accuracy is around 1% in different scenarios after adding randomness α (detailed results are omitted to supplementary). This observation indicates that the high detection performance of *AdvIT* does not rely on very accurate optical flow estimator, which makes the proposed detection widely applicable.
Run-time Analysis For video based tasks, the frame process efficiency is important. Here we show that our adversarial detection approach *AdvIT* processes the videos with minimal overhead, compared with the source frame rate. Theoretically, our approach runs the model inference k times more; while running the model is the major cost, and k is a small constant, we have the same time complexity as the original learner's inference. Empirically, we measure the additional running time for different tasks using an Nvidia 1080Ti GPU. We present the run-time results of model inference, detection, and overhead (subtraction of the two) in Tab. 6.4. For human pose estimation and object detection, *AdvIT* has low overhead, 0.03 and 0.05 seconds respectively, yielding less than two frames of delay. The overhead for segmentation is higher at 0.4s, while the cost to run segmentation on the original input images is much larger, 2.58s on average, and our overhead is only 15.5%. When run in parallel with other real-time action recognition models [96, 98] and flow estimators, our detection pipeline can also achieve close to real-time performance.

Task	Inference	Detection	Overhead
Segmentation	2.58 ± 0.29	2.98 ± 0.27	0.4
Human Pose Estimation	0.02 ± 0.01	0.05 ± 0.01	0.03
Object Detection	0.04 ± 0.01	0.09 ± 0.01	0.05
Action Recognition	0.50 ± 0.01	0.52 ± 0.01	0.02

Table 6.4: Detection overhead of *AdvIT* (in seconds).

6.3 Discussion and Conclusion

We have presented an effective and efficient adversarial frame detection method *AdvIT* for various video based tasks. We have used the state of the art learners, and challenged our detector with the best-known attacks. Further, we have done our best to identify scenarios where an adversary is aware of the detector and may seek to use information about the detector to circumvent it. Even with this strong adaptive adversary we are able to detect nearly all adversarial frames (above 95%). More sophisticated future attacks, which rely on different assumptions than those laid out here may be able to create adversarial frames while fooling our detector, which will be interesting future directions. Given the sequential nature of video, this work indicates that developing new attacks is likely to be more difficult in the video domain than in static image analysis.

Our experiments rely on video tasks where video is taken from continuous sequence. An video that contains "jump cuts" would likely introduce a small number of false positive frames into our detector, as these cuts would be unpredictable by the optical flow algorithm. However, continuous video is consistent with our applications (particularly autonomous driving and surveillance systems), and is unlikely to contain such cuts. It is also possible that we could detect such cuts, as the difference between the pseudo frames and the current frames would still be large, which needs further studies.

It should be noted that the goal of our work is to detects adversarial frames, but is not aimed at remediating or repairing them. Future work could include using pseudo frames as surrogates for suspicious frames or using pseudo frames along with detected adversarial frames to reform attacks.

CHAPTER 7

Conclusion and Future Directions

This thesis focuses on studying the security problem of current DNNs. Starting with the traditional $\mathcal{L}_p$-based threat model, we proposed an algorithm, AdvGAN, to generate $\mathcal{L}_p$-based adversarial examples efficiently by using the generative adversarial network (GAN) in chapter 2. Once trained, the feed-forward generator can produce adversarial perturbations efficiently. When we apply AdvGAN to generate adversarial instances on different models without knowledge of the defenses in place, the generated adversarial examples can attack the state-of-the-art defenses with higher attack success rate than examples generated by the competing methods (FGSM and CW). We believe this property makes AdvGAN a promising candidate for improving adversarial training defense methods, a conjecture supported by some follow-on work, see e.g., Jang et al. [61].

Before moving toward more advanced attacks or defenses under the traditional $\mathcal{L}_p$, we revisited the limitations of traditional $\mathcal{L}_p$-based threat model and provided a rigorous threat model. Based on this, we proposed a new type of perturbation based on spatial transformation, which aims to preserve high perceptual quality for adversarial examples in chapter 2. We have shown that adversarial examples generated by stAdv are more difficult for humans to distinguish from original instances. We also analyzed the attack success rate of these examples under existing defense methods and demonstrate they are harder to defend against, which opens new directions for developing more robust defense algorithms.

Beyond the exploration of adversarial examples in the digital space, we also studied the adversarial behavior in the physical space. We conducted a comprehensive analysis of adversarial behavior from the 3D space by simulating the photo-taken process via a physical renderer in chapter 3. Because the digital images are formed by the processing of illumination, shape and texture, we studied adversarial examples both from texture and shape side. We found that both texture and shape are vulnerable and could be manipulated to generate adversarial examples.

Based on meshAdv, we further explored the adversarial behavior of shape towards the real world industry-level system. We selected the Baidu Apollo LiDAR perceptual module as our targeted system. We designed an approach LiDARadv to generate a printable "adversarial mesh to mislead the LiDAR-based perceptual system and printed the optimized adversarial object to fool the physical

autonomous vehicle in chapter 4. Our results showed that the 3D adversarial meshes could mislead the industry-level machine learning model in real world and caused severer consequences (e.g. crash).

The last part of this thesis aimed at finding solutions to detect adversarial examples in the digital space. We proposed a consistency check based framework to *detect adversarial examples* by exploiting properties inherent in either learning models (in chapter 5) or data (in chapter 6). We proposed to apply spatial consistency information analysis to recognize adversarial examples in segmentation, which has not been considered in either image classification or segmentation as a potential detection mechanism. We showed that such spatial consistency information is different for adversarial and benign instances and can be potentially leveraged to detect adversarial examples even when facing strong adaptive attackers. Additionally, we also showed that we could leverage the property of video data to detect adversarial examples as well. One of the properties of videos is temporal consistency: the next frame is typically the continuation of the objects and scenes in the current frame. We showed that we could leverage the property of temporal consistency to detect adversarial examples among different learning tasks such as segmentation, human pose estimation, and object detection.

Looking ahead, there are several future research directions.

Principled defense against various attacks. Defending against adversarial examples is one of the most important and fundamental problems facing deep learning. Although adversarial training based methods have made significant progress over the past few years to defend against adversarial examples, they were only effective against the small set of L_p-bounded attacks [130] based on our study in this thesis. Looking forward, There are several promising ways to solve this problem as we elaborate next.

1. *Designing diverse distance metrics and general training frameworks.* Beyond L_p norm distance metric, my work, stAdv [130] has proposed the spatial transformed distance metrics, which could be used to improve the diversity of adversarial training based method to defend our attacks. We believe more distance metrics need to be investigated to formalize the threat model of adversarial examples. In addition to more general distance metrics, we also need to make efforts to design general training frameworks to handle various constraints.

2. *Generating diverse data.* AdvGAN [128] achieved the top position in Madry's MNIST challenge as a result of exploring some uncovered regions by using generative adversarial nets (GANs). Therefore, we could explore the capability of GANs to mimic unknown types of attacks instead of just memorizing existing attacks to improve the robustness against various adversarial examples.

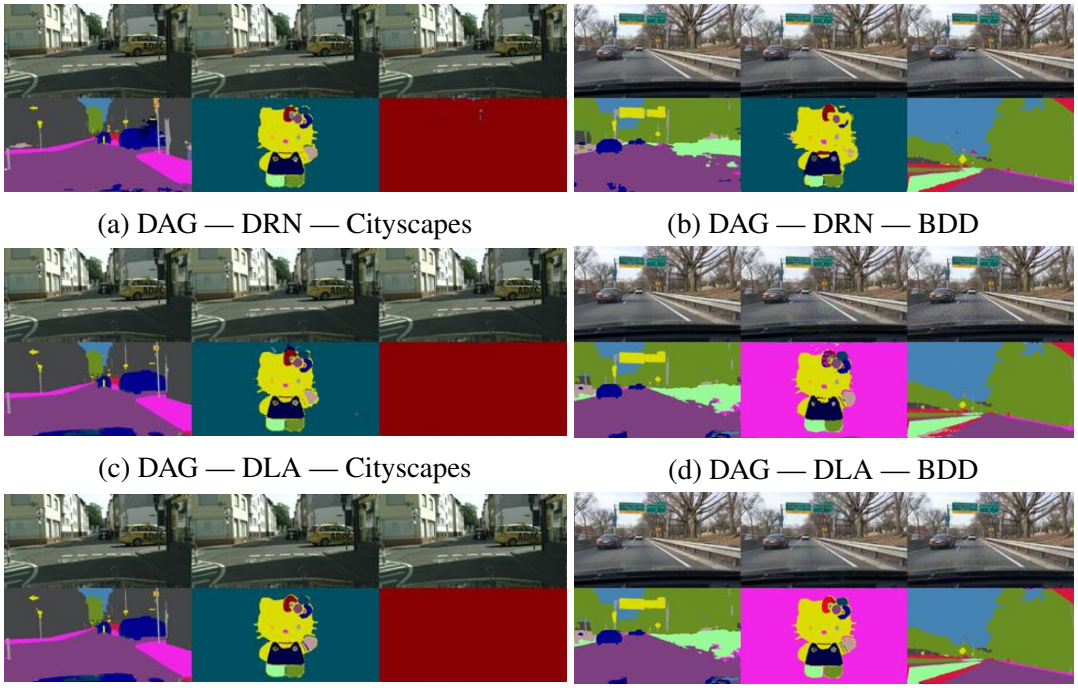
Altruistic adversarial machine learning. Methods designed in the context of adversarial machine learning are generally meant to address the security problem. However, *these methods could potentially benefit other tasks such as image generation, privacy, and generalization*. Recently, [134] showed that the carefully designed adversarial training based method could help to improve the test accuracy of image classification. They achieved the state-of-the-art 85.5% ImageNet top-1 accuracy without extra data. Similar behavior has been discovered in Natural Language Processing domain as well [150]. Therefore, how to broaden the usage of adversarial machine learning in more general contexts is an important future research direction.

Adversarial machine learning inspired by cognitive science. My current research [136] successfully improves the model general robustness via enhancing CNNs to solely use shape information, which motivated by analyzing the robust human visual system. It inspires me to revisit the design of the CNNs, e.g. how to make CNNs mimic human spatial cognition, how to make CNNs truly understand the high-level abstraction and the relations before the decision. On the other hand, researchers in cognitive science have demonstrated the success of computational approaches based on structured, relational information that exhibit human-like learning. It results in successes in terms of requiring few training data and stability. For instance, Kandaswamy et al. [65] leverages Structure Mapping Engine (SME), a computational model of analogical matching a similarity base on Structure Mapping Theory to build AI performance system for a variety of tasks. Although these approaches do not yet achieve the STOA performance compared to deep learning systems, they have shown better data efficiency and stability. Exploring how to combine significant achievements of cognitive science and deep learning model to design a more robust machine learning system is an active research direction in the future.

APPENDIX A

Appendix of Detecting Adversarial Examples by Using the Property of the Learning Model

A.1 Adversarial Examples For Cityscapes and BDD Datasets Against DRN and DLA models



(a) DAG — DRN — Cityscapes (b) DAG — DRN — BDD

(c) DAG — DLA — Cityscapes (d) DAG — DLA — BDD

(e) Houdni — DLA — CityScapes (f) Houdini — DLA — BDD

Figure A.1: Samples of benign and adversarial examples. We use the format "attack method —attack model — dataset" to label the settings of each adversarial examples. Within each subfigure, the first column shows benign images and corresponding segmentation results, the second and third columns show adversarial examples with different adversarial targets (targeting on Kitty/Pure in (a),(c), (d) and on Kitty and Scene in (b),(d),(f)).

Figure A.1 shows the benign and adversarial examples targeting at diverse adversarial targets: "Hello Kitty" (Kitty) and random pure color (Pure) on Cityscapes [28]; and "Hello Kitty" (kitty)

Figure A.2: Attack results of additional targets on Cityscapes. The first column shows benign instance, while 2-4 columns show adversarial examples with target "ECCV 2018", "Remapping", and "Color strip", respectively.

and a real scene without any cars (Scene) on BDD [135] dataset against DRN [138] and DLA [139] segmentation models. In order to increase the diversity of our target set, we also apply different colors for the background of "Hello Kitty" on BDD dataset against DLA model.

Figure A.2 shows the additional adversarial targets, including "ECCV 2018", "Remapping", and "Color strip". Here remapping means we generate an adversarial target by shifting the numerical label of each class in the ground truth by a constant offset. This way, we can guarantee that each target has no overlap with the ground truth mask. For "Color strip", we divide the target into 19 strips evenly, each of which is filled with a class label, aiming to mitigate possible bias for different classes.

A.2 Spatial Consistency Based Method

A.2.1 Spatial Context Analysis

Algorithm 1 describes the algorithm of **getOverlapPatches**. Figure A.3 shows the heatmaps of the per-pixel self-entropy on Cityscapes and BDD dataset against DRN and DLA models. It is clearly shown that the adversarial instances have higher entropy than benign ones. Table A.1 shows that the detection results (AUC) based on spatial consistency method with fix patch size. It demonstrates that the spatial consistency information can help to detect adversarial examples with AUC nearly 100% on BDD dataset. Table A.4 A.5 show the results on additinal targets on Cityscapes and BDD datasets. Table A.2 A.3 show the detection results (AUC) based on spatial consistency method with random patch size. They show that random patch sizes achieve the similar detection result.

Method	Model	mIOU	Detection				Detection Adap			
			DAG		Houdini		DAG		Houdini	
			Scene	Kitty	Scene	Kitty	Scene	Kitty	Scene	Kitty
Scale (std=0.5)	DRN (16.4M)	54.5	96%	100%	99%	100%	69%	89%	46%	91%
Scale (std=3.0)			100%	100%	100%	100%	31%	89%	1%	48%
Scale (std=5.0)			100%	100%	100%	100%	8%	84%	0%	36%
Scale (std=0.5)	DLA (18.1M)	46.29	96%	88%	99%	99%	89%	90%	80%	58%
Scale (std=3.0)			100%	100%	100%	100%	66%	88%	11%	26%
Scale (std=5.0)			98%	100%	99%	100%	32%	78%	2%	12%
Spatial (K=1)	DRN (16.4M)	54.5	98%	100%	99%	99%	89%	99%	89%	99%
Spatial (K=5)			100%	100%	100%	100%	100%	100%	100%	100%
Spatial (K=10)			100%	100%	100%	100%	100%	100%	100%	100%
Spatial (K=50)			100%	100%	100%	100%	99%	100%	99%	100%
Spatial (K=1)	DLA (18.1M)	46.29	98%	99%	95%	95%	96%	99%	98%	95%
Spatial (K=5)			100%	100%	98%	98%	99%	100%	99%	96%
Spatial (K=10)			100%	100%	99%	99%	99%	100%	99%	96%
Spatial (K=50)			100%	100%	99%	99%	100%	100%	99%	93%

Table A.1: Detection results (AUC) of image spatial (Spatial) and scale consistency (Scale) based methods on BDD dataset. The number in parentheses of the "Model" shows the number of parameters for the target mode, and "mIOU" shows the performance of segmentation model on pristine data. We color all the AUC less than 80% with red.

A.2.2 Scale Consistency Analysis

We applied image scaling to the adversarial examples generated by Houdini [27] and DAG [132] on Cityscapes and BDD datasets against DRN and DLA models. The result shows in Figure A.4. We can find the same phenomenon that when we applying Gaussian blurring with high std (3 and 4), adversarial perturbation is harmed and segmentation result are no longer adversarial targets.

Table A.1 shows that the method based on image scale information can achieve similarly AUC compared with spatial consistency based method on BDD.

Method (Spatial)	Model	mIOU	Detection				Detection Adap			
			DAG		Houdini		DAG		Houdini	
			Pure	Kitty	Pure	Kitty	Pure	Kitty	Pure	Kitty
K=1	DRN (16.4M)	66.7	$91 \pm 0.1\%$	$88 \pm 0.1\%$	$91 \pm 0.3\%$	$90 \pm 0.1\%$	$97 \pm 0\%$	$92 \pm 0.1\%$	$90 \pm 2.5\%$	$93 \pm 0.1\%$
K=5			$99 \pm 0.1\%$	$99 \pm 0.1\%$	$100 \pm 0.0\%$	$99 \pm 0.1\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$
K=10			$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$
K=50			$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$
K=1	DLA (18.1M)	74.5	$96 \pm 0.1\%$	$98 \pm 0.1\%$	$96 \pm 0.1\%$	$96 \pm 0.1\%$	$99 \pm 0.3\%$	$99 \pm 0.1\%$	$98 \pm 0.4\%$	$99 \pm 0.1\%$
K=5			$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$
K=10			$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$
K=50			$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$

Table A.2: Detection results (AUC) of image spatial (Spatial) based method with random patch size on Cityscapes dataset.

Method (Spatial)	Model	mIOU	Detection				Detection Adap			
			DAG		Houdini		DAG		Houdini	
			Pure	Kitty	Pure	Kitty	Pure	Kitty	Pure	Kitty
K=1	DRN (16.4M)	54.5	$91 \pm 0.1\%$	$88 \pm 0.1\%$	$91 \pm 0.3\%$	$90 \pm 0.1\%$	$97 \pm 0\%$	$92 \pm 0.1\%$	$90 \pm 2.5\%$	$93 \pm 0.1\%$
K=5			$99 \pm 0.1\%$	$99 \pm 0.1\%$	$100 \pm 0.0\%$	$99 \pm 0.1\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$
K=10			$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$
K=50			$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$
K=1	DLA (18.1M)	46.29	$98 \pm 0\%$	$96 \pm 0.1\%$	$95 \pm 0.1\%$	$92 \pm 0\%$	$98 \pm 0.1\%$	$94 \pm 0\%$	$97 \pm 0.1\%$	$90 \pm 0\%$
K=5			$100 \pm 0\%$	$99 \pm 0\%$	$99 \pm 0\%$	$98 \pm 0\%$	$99 \pm 0\%$	$98 \pm 0\%$	$99 \pm 0\%$	$90 \pm 0\%$
K=10			$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$98 \pm 0\%$	$100 \pm 0\%$	$99 \pm 0\%$	$100 \pm 0\%$	$96 \pm 0\%$
K=50			$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$99 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$	$100 \pm 0\%$

Table A.3: Detection results (AUC) of image spatial (Spatial) based method with random patch size on BDD dataset.

Method (Spatial)	Model	mIOU	Detection						Detection Adap					
			DAG			Houdini			DAG			Houdini		
			ECCV	Remap	Strip	ECCV	Remap	Strip	ECCV	Remap	Strip	ECCV	Remap	Strip
K=1	DRN (16.4M)	66.7	93%	91%	91%	91%	91%	91%	90%	92%	89%	90%	92%	90%
K=5			99%	100%	99%	99%	100%	99%	100%	99%	100%	100%	100%	99%
K=10			100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	99%
K=50			100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
K=1	DLA (18.1M)	74.5	96%	99%	97%	95%	97%	96%	99%	99%	98%	99%	99%	98%
K=5			100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
K=10			100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	99%
K=50			100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

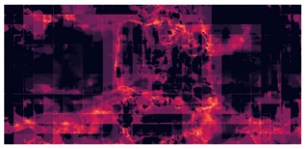
Table A.4: Detection results (AUC) of spatial consistency (Spatial) based method on Cityscapes dataset for additional targets.

Method (Spatial)	Model	mIOU	Detection						Detection Adap					
			DAG			Houdini			DAG			Houdini		
			ECCV	Remap	Strip	ECCV	Remap	Strip	ECCV	Remap	Strip	ECCV	Remap	Strip
K=1	DRN (16.4M)	54.5	99%	99%	99%	99%	99%	99%	99%	99%	99%	99%	98%	97%
K=5			100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	98%
K=10			100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	99%	97%
K=50			100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	99%	97%
K=1	DLA (18.1M)	46.29	99%	99%	99%	98%	97%	98%	99%	99%	99%	98%	97%	99%
K=5			100%	100%	100%	100%	99%	99%	100%	100%	100%	99%	99%	99%
K=10			100%	100%	100%	100%	100%	100%	100%	100%	100%	99%	99%	99%
K=50			100%	100%	100%	100%	100%	100%	100%	100%	100%	99%	99%	99%

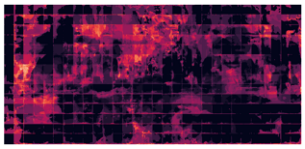
Table A.5: Detection results (AUC) of spatial consistency (Spatial) based method on BDD dataset for additional targets.

(a) Original example — DLA — Cityscapes

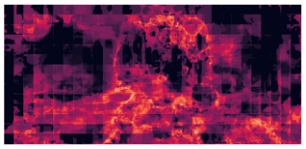
(b) Benign example — DLA — Cityscapes



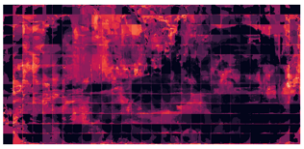
(c) DAG — Kitty — DLA — Cityscapes



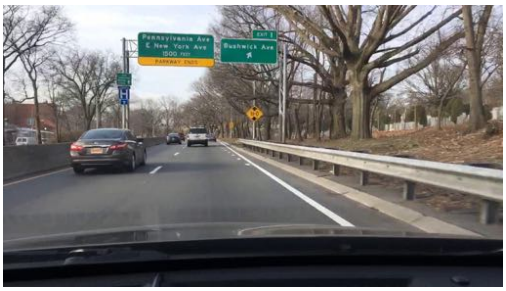
(d) DAG — Pure — DLA — Cityscapes



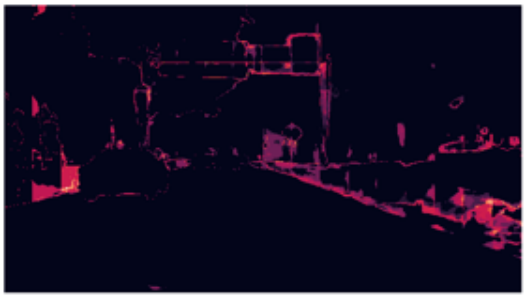
(e) Houdini — Kitty — DLA — Cityscapes



(f) Houdini — Pure — DLA — Cityscapes

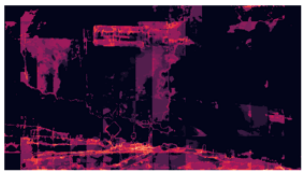


(g) Original example — DRN — BDD

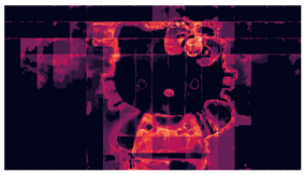


(h) Benign example — DRN — BDD

(i) DAG — Kitty — DRN — BDD

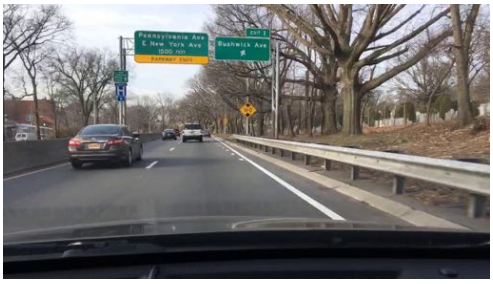


(j) DAG — Scene — DRN — BDD

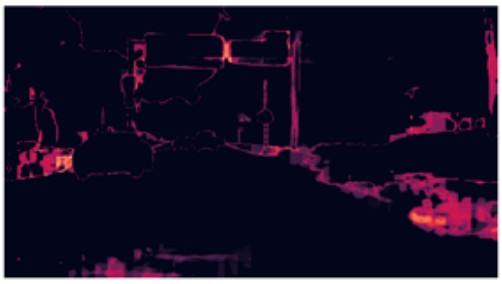


(k) Houdini — Kitty — DRN — BDD

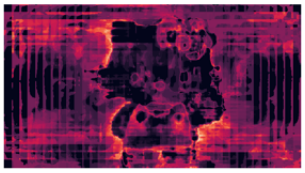
(l) Houdini — Scene — DRN — BDD



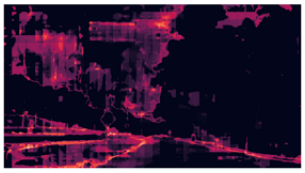
(m) Original example—DLA—BDD



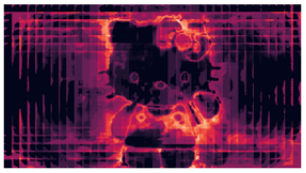
(n) Benign example — DLA — BDD



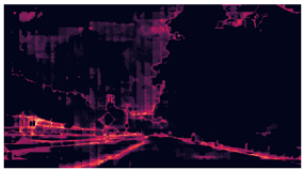
(o) DAG — Kitty — DLA — BDD



(p) DAG — Scene — DLA — BDD

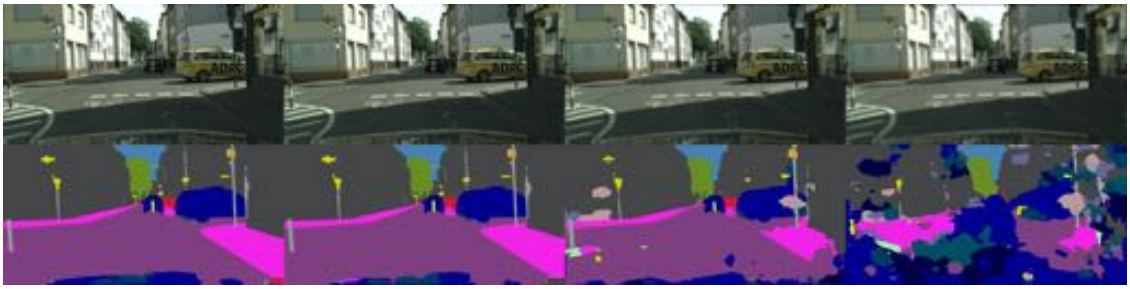


(q) Houdini — Kitty — DLA — BDD



(r) Houdini — Real — DLA — BDD

Figure A.3: Heatmap of per-pixel self-entropy. (a), (b), (g), (h), (m) and (n) show benign images and its corresponding per-pixel self-entropy heatmaps. We use the format "examples — attack model — dataset" to label them. For the rest, we use the format "attack method — target label — attack model — dataset" to label each subcaption.



(a) Benign example — DLA — Cityscapes

(b) DAG — Kitty — DLA — Cityscapes

(c) DAG — Pure — DLA — Cityscapes

(d) Houdini — Kitty — DLA — Cityscapes

(e) Houdini — Pure — DLA — Cityscapes

(f) Benign example — DRN — BDD

(g) DAG — Kitty — DRN — BDD

(h) DAG — Scene — DRN — BDD

(i) Houdini — Kitty — DRN — BDD

(j) Houdini — Scene — DRN — BDD

(k) Benign example — DLA — BDD

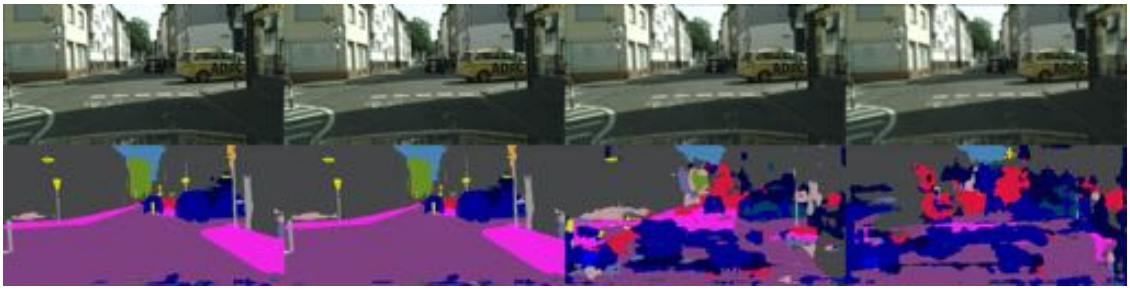
(l) DAG — Kitty — DLA — BDD

(m) DAG — Scene — DLA — BDD

(n) Houdini — Kitty — DLA — BDD

(o) Houdini — Scene — DLA — BDD

Figure A.4: Examples of images and corresponding segmentation results before/after image scaling. For each subfigure, the first column shows benign/adversarial images, while the following columns represent images after scaling by applying Gaussian kernel with std as 0.5, 3, and 5, respectively. (a),(f) and (k) show benign images before/after image scaling and the corresponding segmentation results and we use the format "example — attack model — dataset" to identify the corresponding model and dataset; (b)-(e), (g)-(j) and (l)-(o) present similar results for adversarial images and we use the format "attack method — target label — attack model — dataset" to label the settings of each image.



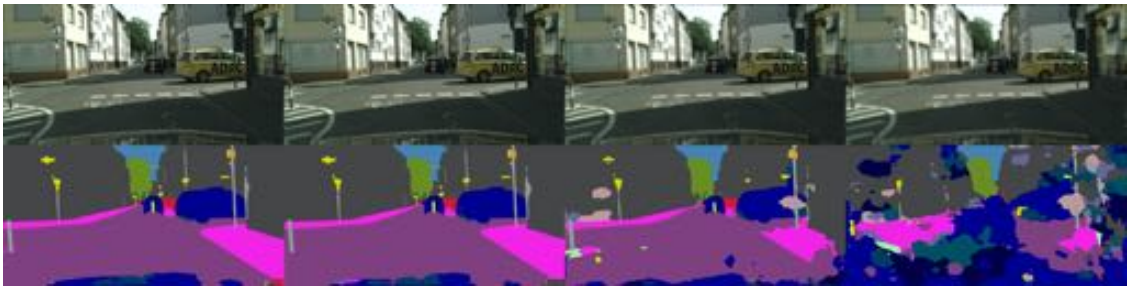
(a) Benign example — DRN — Cityscapes

(b) DAG — Kitty — DRN — Cityscapes

(c) DAG — Pure — DRN — Cityscapes

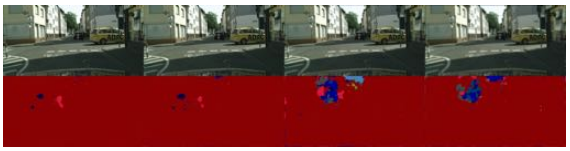
(d) Houdini — Kitty — DRN — Cityscapes

(e) Houdini — Pure — DRN — Cityscapes



(f) Benign example — DLA — Cityscapes

(g) DAG — Kitty — DLA — Cityscapes



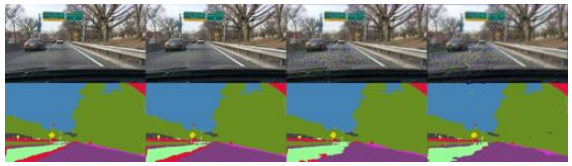
(h) DAG — Pure — DLA — Cityscapes

(i) Houdini — Kitty — DLA — Cityscapes

(j) Houdini — Pure — DLA — Cityscapes

(k) Benign example — DRN — BDD

(l) DAG — Kitty — DRN — BDD



(m) DAG — Scene — DRN — BDD

(n) Houdini — Kitty — DRN — BDD

(o) Houdini — Scene — DRN — BDD

(p) Benign example — DLA — BDD

(q) DAG — Kitty — DLA — BDD

(r) DAG — Scene — DLA — BDD

(s) Houdini — Kitty — DLA — BDD

(t) Houdini — Scene — DLA — BDD

Figure A.5: Examples of images and corresponding segmentation results for adaptive attack against image scaling. For each subfigure, the first column shows benign/adversarial images, while the following columns show images after scaling by applying Gaussian kernel with std as 0.5, 3, and 5, respectively. (a), (f), (k) and (p) show benign images before/after image scaling and the corresponding segmentation results and The format "example — attack model — dataset" uses to identify the corresponding model and dataset; (b)-(e), (g)-(j), (l)-(o) and (q)-(t) present similar results for adaptive adversarial images and we describe by the format "attack method — target label — attack model — dataset".

A.2.3 Adaptive Attack Evaluation

Algorithm 4: DAG adaptive attack against spatial consistency method

input: Input image $\mathbf{X}$;
Number of attack patches K;
Patch size s;
Segmentation model f;
L^2 bound $\mathbf{b}$;
Recognition targets $\mathcal{T} = \{t_1, t_2, \cdots, t_N\}$;
Adversarial label set $\mathbf{Y} = \{Y_{t_1}, Y_{t_2}, \cdots, Y_{t_N}\}$;
Maximal iteration M_0 ;

output: Adversarial perturbation $\mathbf{r}$;

Initialization :$\mathbf{X}_0 \leftarrow \mathbf{X}, \mathbf{r} \leftarrow 0, m \leftarrow 0, \mathcal{T}_0 \leftarrow \mathbf{Y}, w = \mathbf{X}.width, h = \mathbf{X}.height$;

while $m < M_0$ & $\mathbf{l2norm}(\mathbf{r}) < \mathbf{b}$ **do**
  $\mathcal{T}_m = \{t_n | \text{argmax}_c\{f_c(\mathbf{X}_m, t_n)\} \neq Y_{t_n}\}$;
  $\mathbf{r_m} \leftarrow \sum_{t_n \in \mathcal{T}_m} \nabla_{\mathbf{x}_m} f_{Y_{t_n}}(\mathbf{X}_m, t_n)$;
  /* Attack C random patches */;
  for $k \leftarrow 0$ **to** K **do**
    Generate two random integers I_1, I_2 where $0 < I_1 < w - s$ and $0 < I_2 < h - s$;
    $P_k = \mathbf{X}[I_1 : I_1 + s, I_2 : I_2 + s]$;
    $\mathcal{T}_k = \{t_{n'} | \text{argmax}_c\{f_c(P_k, t_{n'})\} \neq Y_{t_{n'}} \ \& \ t_{n'} \in \mathcal{T}_{\Updownarrow}\}$;
    $\mathbf{r_m} \overset{+}{\leftarrow} \sum_{t_{n'} \in \mathcal{T}_k} \nabla_{\mathbf{P_k}} f_{Y_{t_{n'}}}(P_k, t_{n'})$
  end
  $\mathbf{r'}_m \leftarrow \frac{\gamma}{\|\mathbf{r}_m\|_\infty} \mathbf{r}_m$;
  $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{r'}_m$;
  $\mathbf{X}_{m+1} \leftarrow \mathbf{X}_m + \mathbf{r'}_m$;
  $m \leftarrow m + 1$;
end

Return: $\mathbf{r}$

Here we illustrate the adaptive attack algorithm based on DAG and Houdini against spatial consistency method in Algorithm 4 and Algorithm 5. Instead of only attacking the benign image, the adaptive attack here will randomly pick some patches and attack them with the whole image together.

Let $\mathbf{X}$ be an image and K is the number of patches selected from $\mathbf{X}$ to perform adaptive attack. We define $\mathcal{T} = \{t_1, t_2, \cdots, t_N\}$ as the set comprising the coordinates of the recognition target pixels. f denotes the segmentation network, and we use $f(\mathbf{X})$ to denotes the classification score of the entire image and $f(\mathbf{X}, t_n)$ to denote the classification score vector at pixel n. $\mathbf{Y} = Y_{t_1}, Y_{t_2}, \cdots, Y_{t_N}$ denotes the adversarial label set where Y_{t_n} represents the adversarial label of pixel n. Given an input tensor $\mathbf{r} \in \mathbb{R}^{w \times h \times c}$, the function returns its L^2 norm defined to be $||\mathbf{r}||_2 = \sqrt{\frac{\sum_{i=1}^{w} \sum_{j=1}^{h} \sum_{k=1}^{c} \mathbf{r}_{ijk}^2}{w \cdot h \cdot c}}$. In

Algorithm 5: Houdini adaptive attack against spatial consistency method

input: Input image $\mathbf{X}$;
Number of attack patches K;
Patch size s;
Segmentation model f;
L^2 bound $\mathbf{b}$;
Adversarial target label $\mathbf{Y}$;
Maximal iteration M_0 ;
Houdini loss $\mathbf{H}$;

output: Adversarial perturbation $\mathbf{r}$;

Initialization :$\mathbf{X}_0 \leftarrow \mathbf{X}, \mathbf{r} \leftarrow 0, m \leftarrow 0, \mathcal{T}_r \leftarrow \mathbf{Y}, w \leftarrow \mathbf{X}.width, h \leftarrow \mathbf{X}.height$;

while $m < M_0$ **do**
  $\mathbf{r_m} \leftarrow 0$;
  `/* get the gradient of the perturbation from the objective */`;
  $\mathbf{r_m} \leftarrow \mathbf{r_m} + \nabla_{\mathbf{r}} \mathbf{H}(f(\mathbf{X} + \mathbf{r}), \mathbf{Y})$;
  $\mathbf{r_m} \leftarrow \mathbf{r_m} + \nabla_{\mathbf{r}} \max(\mathbf{l2norm}(\mathbf{r}) - b, 0)$;
  for $k \leftarrow 0$ **to** K **do**
    Generate two random interger numbers I_1, I_2 where $0 < I_1 < w - s, 0 < I_2 < h - s$;
    $\mathbf{r}_m \leftarrow \mathbf{r}_m + \nabla_{\mathbf{r}} \mathbf{H}(f((\mathbf{X} + \mathbf{r})[i : i + s, j : j + s]), \mathbf{Y}[i : i + s, j : j + s])$;
  end
  $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{r}_m$;
end

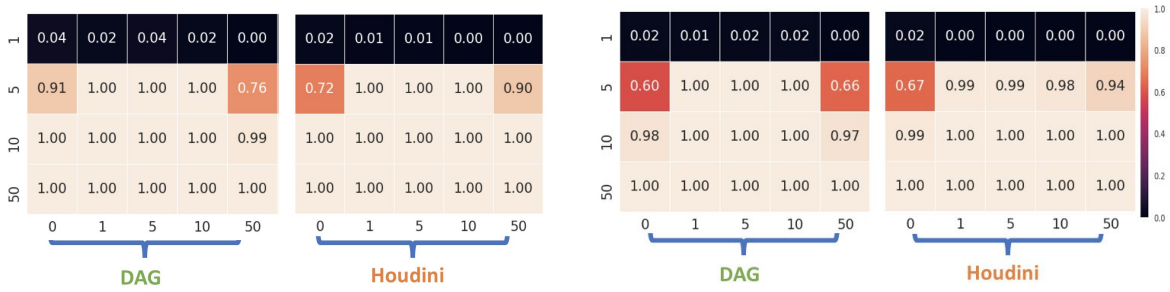
Return: $\mathbf{r}$

Algorithm 5, we follow the same definition of Houdini loss H as proposed in the work [27]. We set the maximal number of iteration to be approximately 300[1] in all settings. We set L^2 bound to be 0.06 for simplicity.

The detection results in term of AUC of the spatial consistency based method against such adaptive attacks on BDD and Cityscapes are shown in Table A.1 A.2 A.3 A.4 A.5. Even against such strong adaptive attacks, the spatial consistency based method can still achieve nearly 100% AUC. Figure A.6 shows the confusion matrix of detection result for adversaries and detection method choosing various K. It is clear that when we choose $K = 50$, it is already sufficient to detect sophisticated attacks on Cityscapes dataset against DLA model and can also achieved 100% detection rate with false positive rate 95% on BDD.

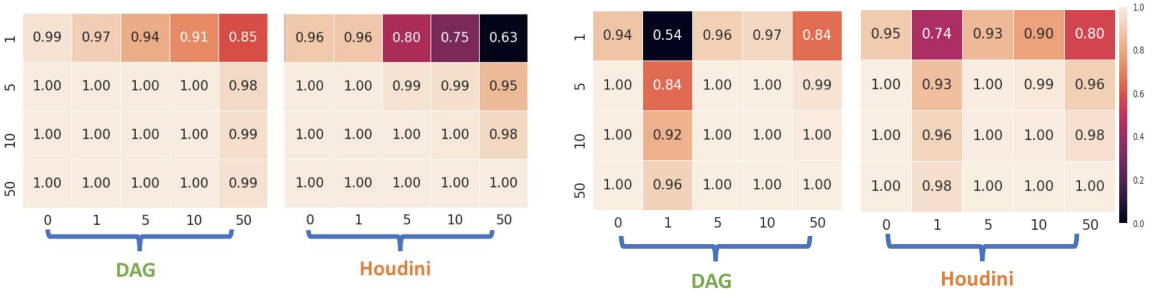
The detection results of the image scaling based method against adaptive attacks on BDD are shown in Table A.1. For image scaling based detection method, it can be easily attacked (AUC drops drastically). Figure A.5 shows the qualitative results. It is obvious that even under Gaussian

[1] 300 is approximately three times the average number of iterations in non-adaptive attack.



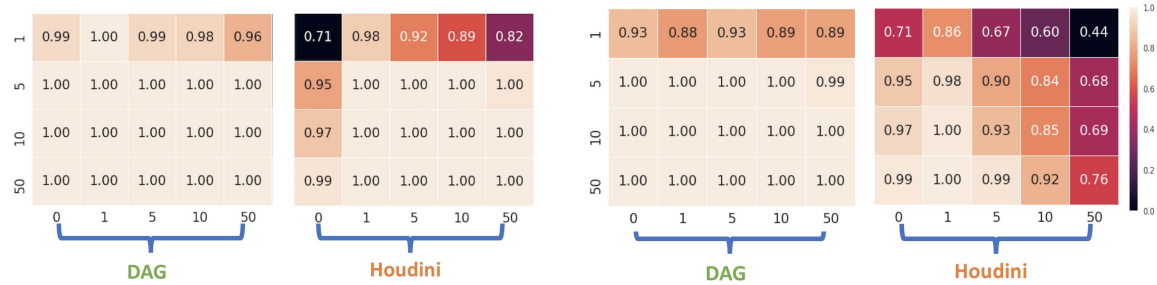
(a) DLA — Kitty — Cityscapes

(b) DLA — Pure — Citysacpes



(c) DRN — Kitty — BDD

(d) DRN — Scene — BDD



(e) DLA — Kitty — BDD

(f) DLA — Scene — BDD

Figure A.6: Detection performance of spatial consistency based method against adaptive attack with different K. We use the format "attack model — target label — dataset" to label the settings for each figure. X-axis indicates the number of patches selected to perform the adaptive attack (0 means regular attack). Y-axis indicates the number of overlapping regions selected during detection. We select the minimal mIOU from benign patches as our threshold on Cityscapes, and the one which guarantees accuracy as above 95% on benign images for BDD.

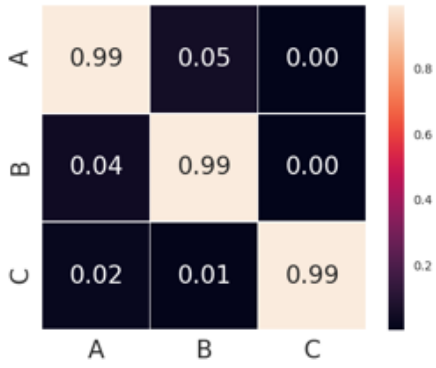
kernel with large standard deviation, the adversarial example can still be fooled into predicting different malicious targets ("Kitty", "Pure", "Scene") on Cityscapes and BDD dataset against DRN and DLA models.

A.3 Additional Results for Transferability Analysis

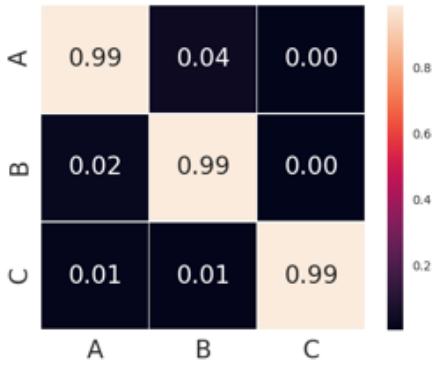
We present additional results for transferability analysis in Figure A.7 to Figure A.11.

Figure A.7 to Figure A.8 show the transferability analysis results for segmentation models. We report pixel-wise attack success rate for the pure target and normalized mIoU after eliminating K classes with the lowest IoU values for other targets. We set K to be 13 for CityScapes dataset and 5 for BDD dataset. Additional qualitative results are presented in Figure A.9 to Figure A.10.

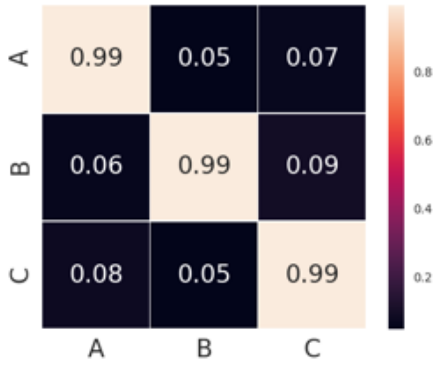
Fig. A.11 shows the transferability experiments results for classification models under targeted attack. The adversarial images are generated using iterative FGSM method from MNIST and CIFAR10 datasets. The caption of each sub-figure indicates the dataset and the attacked classification model.



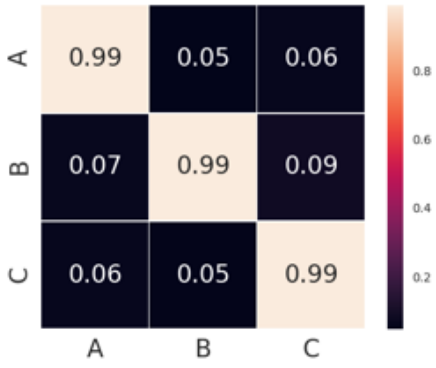
(a) DAG | Pure | DRN-D-22



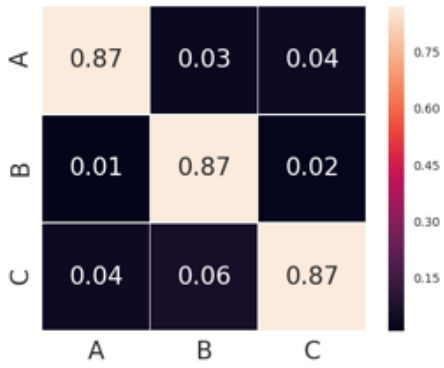
(b) Houdini | Pure | DRN-D-22



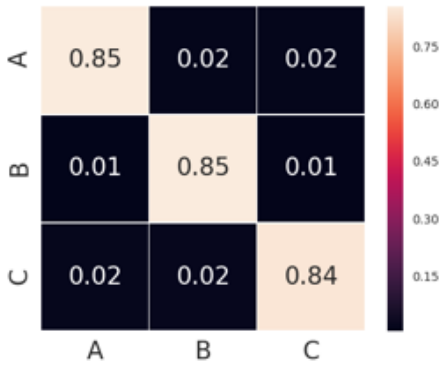
(c) DAG | Pure | DRN-C-26



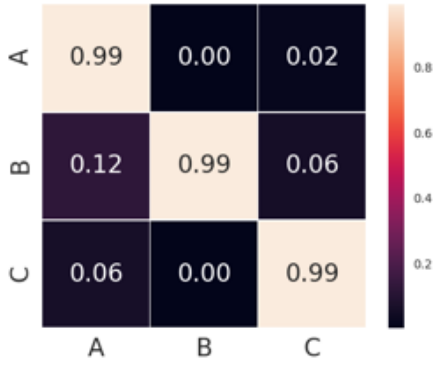
(d) Houdini | Pure | DRN-C-26



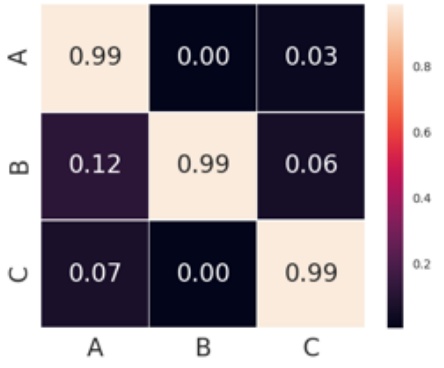
(e) DAG | Hello Kitty | DRN-C-26



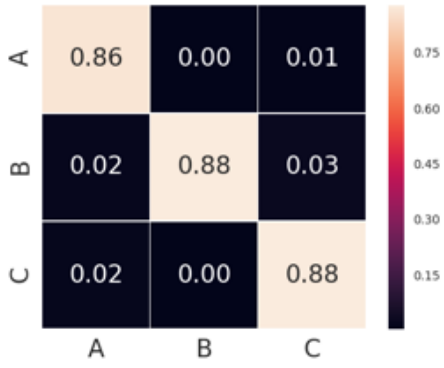
(f) Houdini | Hello Kitty | DRN-C-26



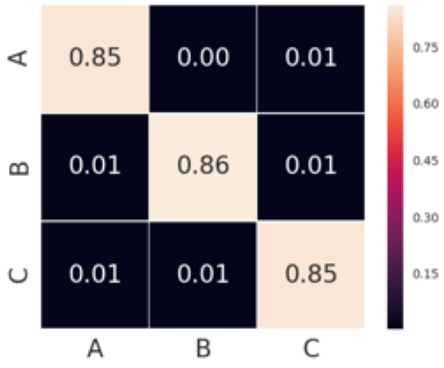
(g) DAG | Pure | DLA34UP



(h) Houdini | Pure | DLA34UP

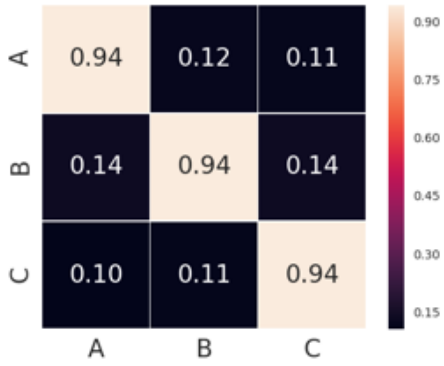


(i) DAG | Hello Kitty | DLA34UP

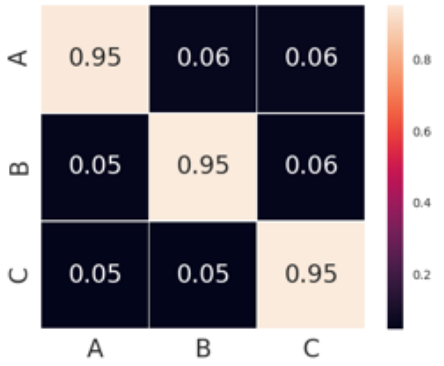


(j) Houdini | Hello Kitty | DLA34UP

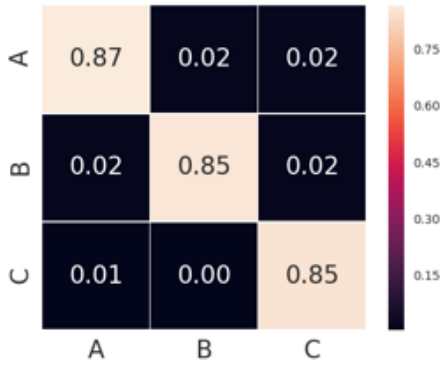
Figure A.7: Transferability analysis on CityScapes dataset: cell (i, j) shows the normalized mIoU value or pixel-wise attack success rate of adversarial examples generated against model j and evaluate on model i. Model A,B,C have the same architecture (DRN-C-26 or DLA34UP) with different initialization. We use format "attack method | attack target | model " to denote the caption of each sub-figure.



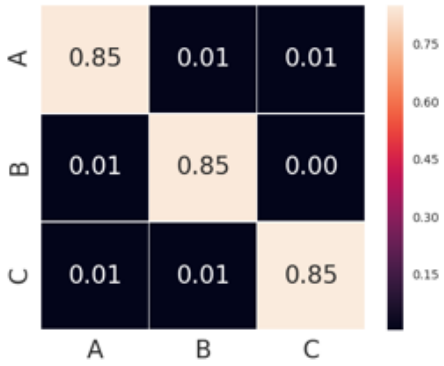
(a) DAG | Scene | DRN-D-22



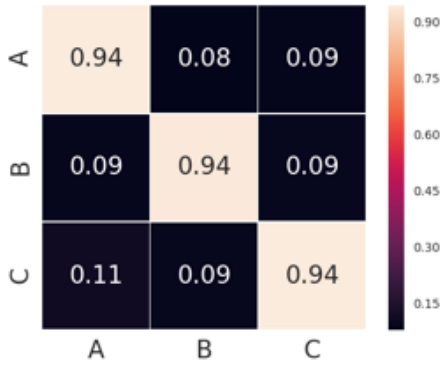
(b) Houdini | Scene | DRN-D-22



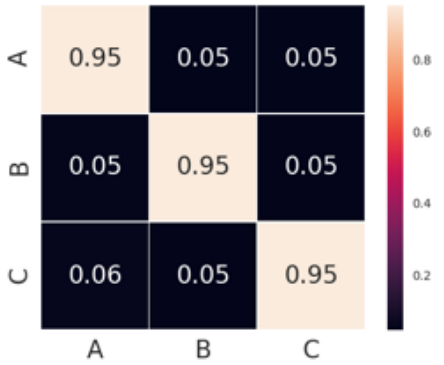
(c) DAG | Hello Kitty | DRN-D-22



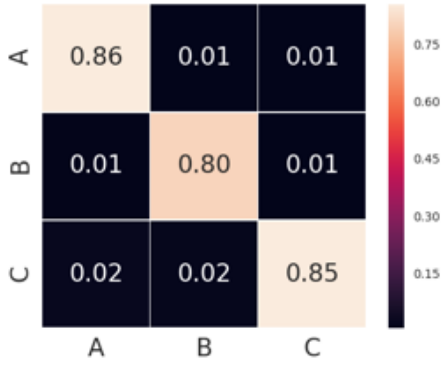
(d) Houdini | Hello Kitty | DRN-D-22



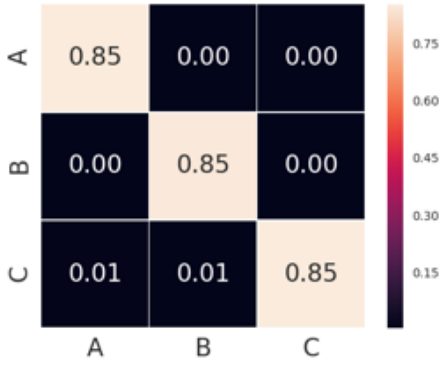
(e) DAG | Scene | DRN-C-26



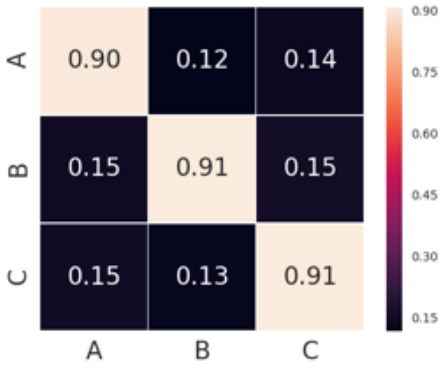
(f) Houdini | Scene | DRN-C-26



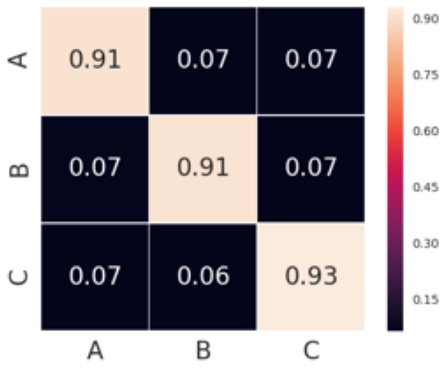
(g) DAG | Hello Kitty | DRN-C-26



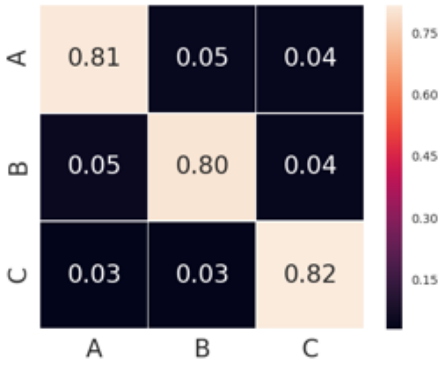
(h) Houdini | Hello Kitty | DRN-C-26



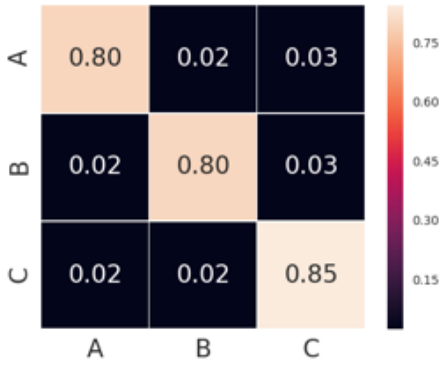
(i) DAG | Scene | DLA34UP



(j) Houdini | Scene | DLA34UP

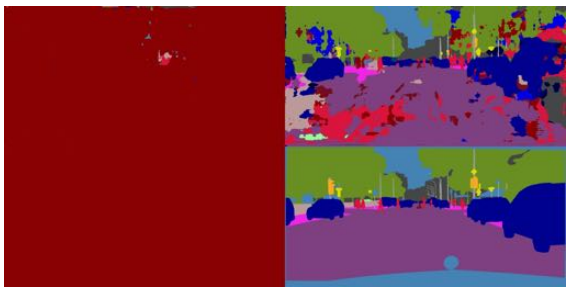


(k) DAG | Hello Kitty | DLA34UP

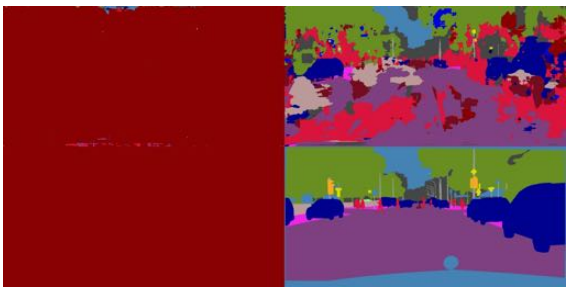


(l) Houdini | Hello Kitty | DLA34UP

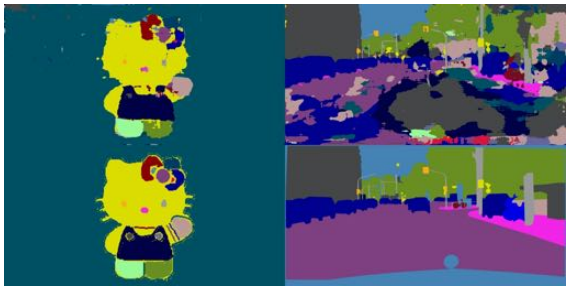
Figure A.8: Transferability analysis on BDD dataset.



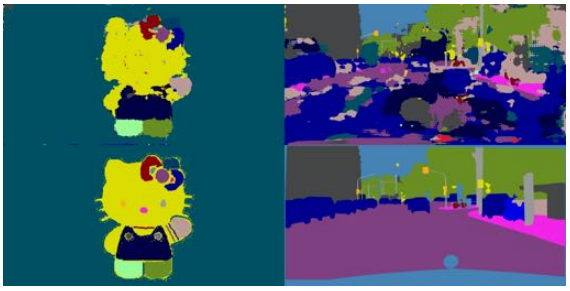
(a) DAG | Pure | DRN-D-22



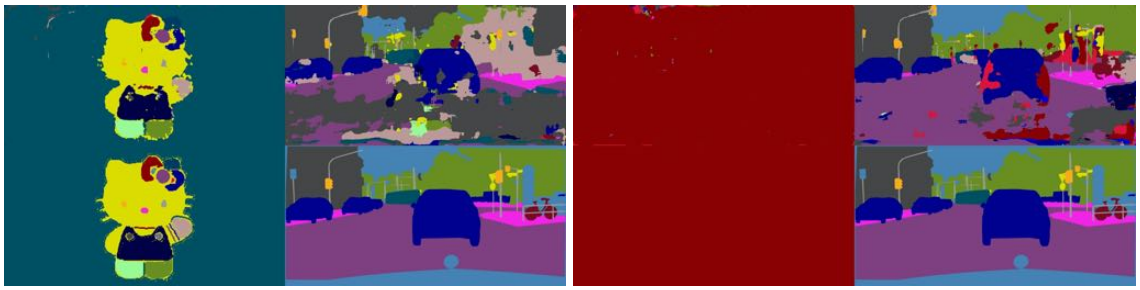
(b) Houdini | Pure | DRN-D-22



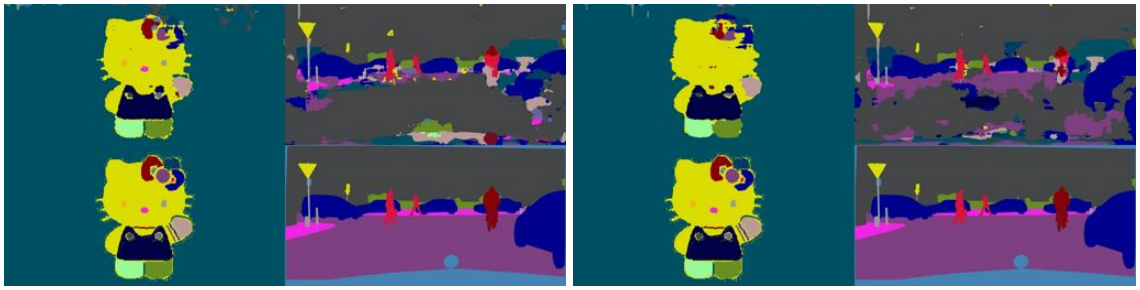
(c) DAG | Hello Kitty | DRN-D-22



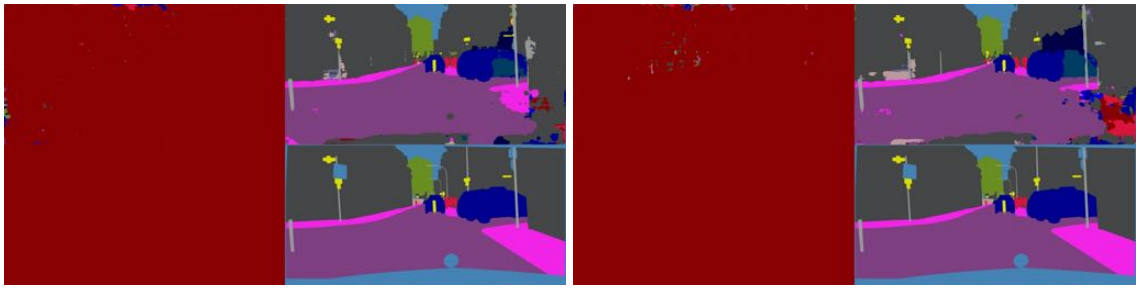
(d) Houdini | Hello Kitty | DRN-D-22



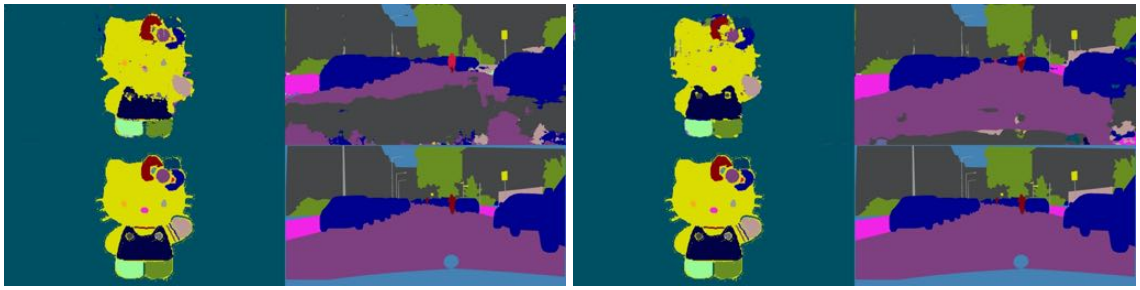
(e) DAG | Pure Rider | DRN-C-26 (f) Houdini | Pure Rider | DRN-C-26



(g) DAG | Hello Kitty | DRN-C-26 (h) Houdini | Hello Kitty | DRN-C-26

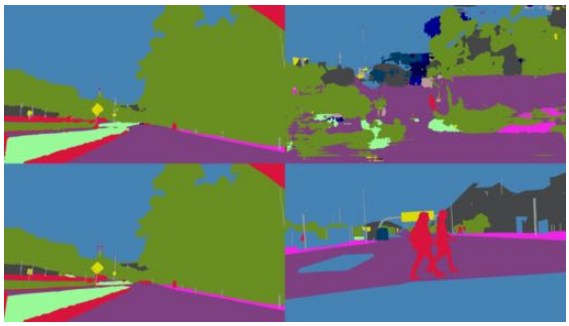


(i) DAG | Pure Rider |DLA34UP (j) Houdini | Pure Rider |DLA34UP

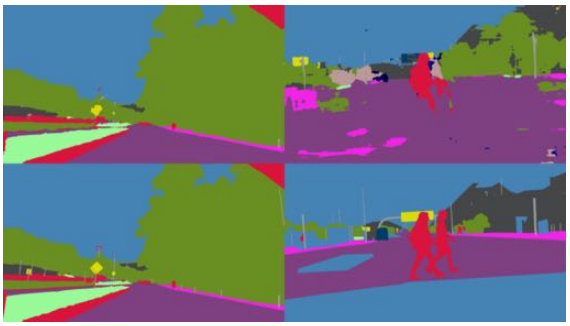


(k) DAG | Hello Kitty |DLA34UP (l) Houdini | Hello Kitty |DLA34UP

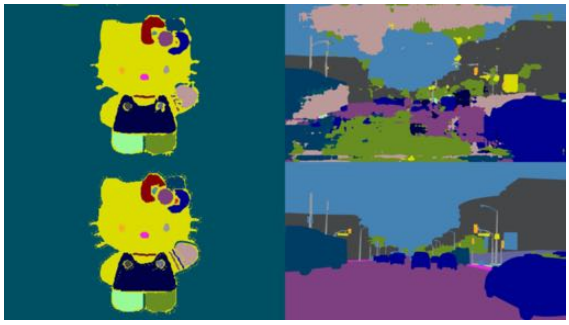
Figure A.9: Transferability visualization on CityScapes dataset. In each sub-figure, the first row presents the segmentation results of adversarial example on model A (targeted model) and model B. The second row shows the adversarial target and the ground truth. We use format "attack method | attack target | model " to denote the caption of each sub-figure.



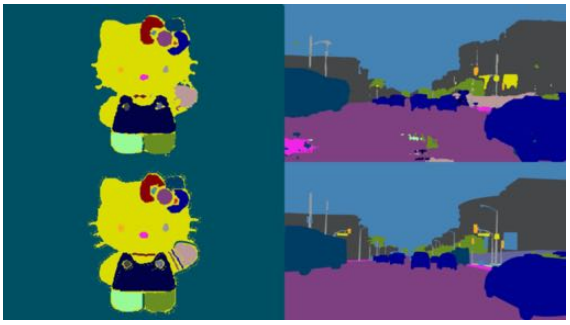
(a) DAG | Scene | DRN-D-22



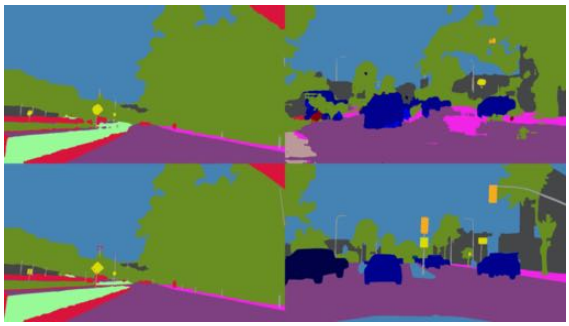
(b) Houdini | Scene | DRN-D-22



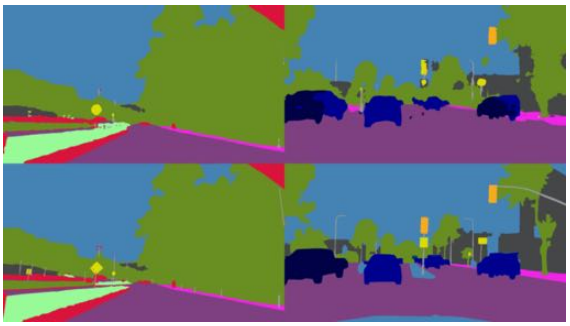
(c) DAG | Hello Kitty | DRN-D-22



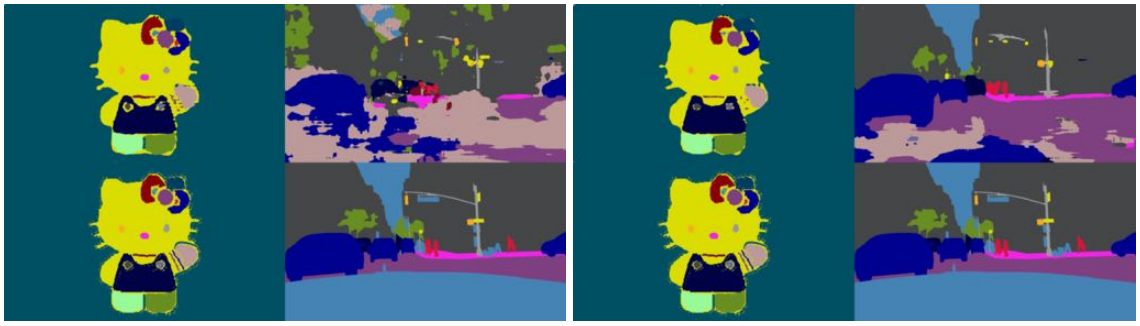
(d) Houdini | Hello Kitty | DRN-D-22



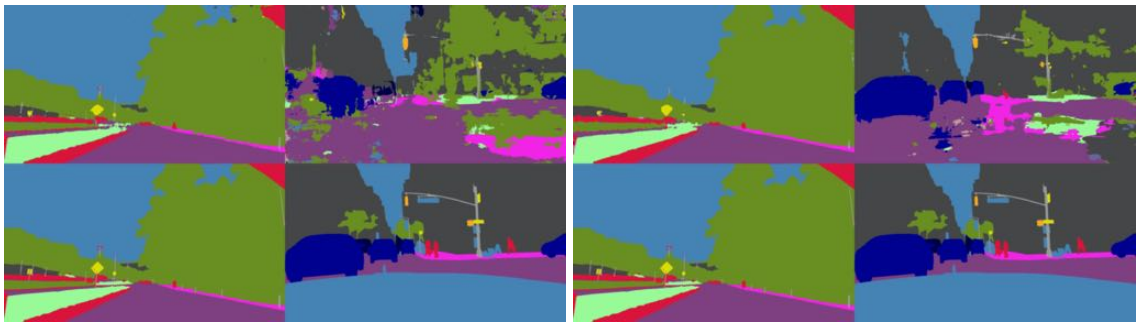
(e) DAG | Scene | DRN-C-26



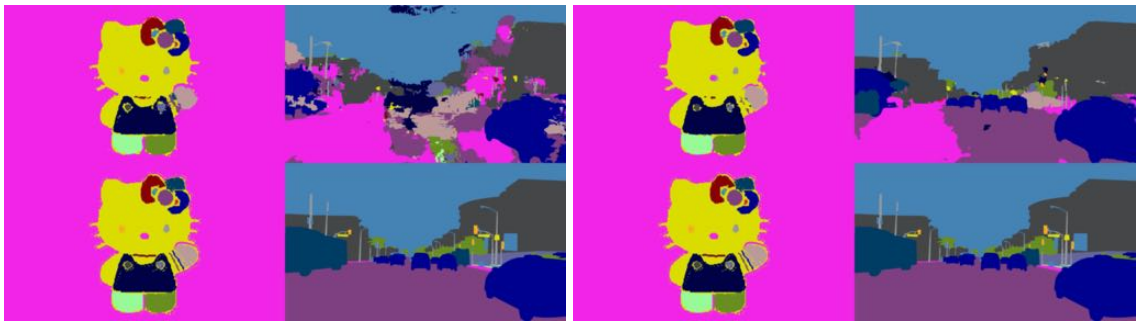
(f) Houdini | Scene | DRN-C-26



(g) DAG | Hello Kitty | DRN-C-26 (h) Houdini | Hello Kitty | DRN-C-26

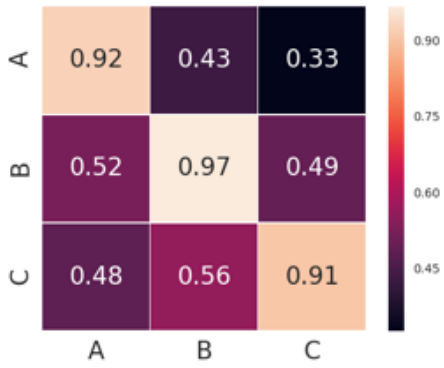


(i) DAG | Scene | DLA34UP (j) Houdini | Scene | DLA34UP

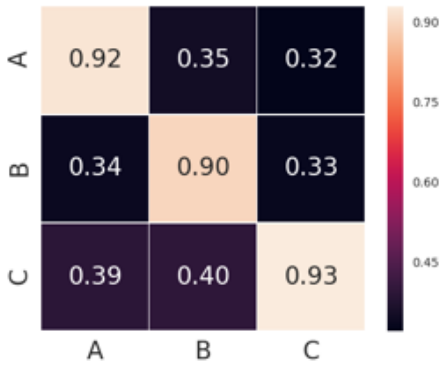


(k) DAG | Hello Kitty | DLA34UP (l) Houdini | Hello Kitty | DLA34UP

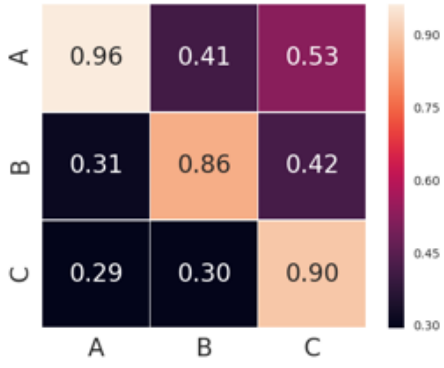
Figure A.10: Transferability visualization on BDD dataset.



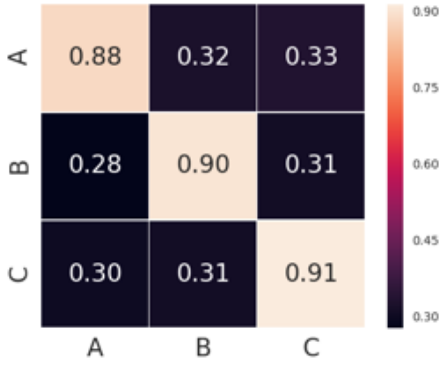
(a) MNIST | DRN-D-22



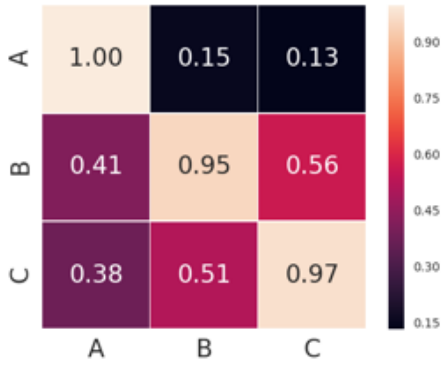
(b) CIFAR10 | DRN-D-22



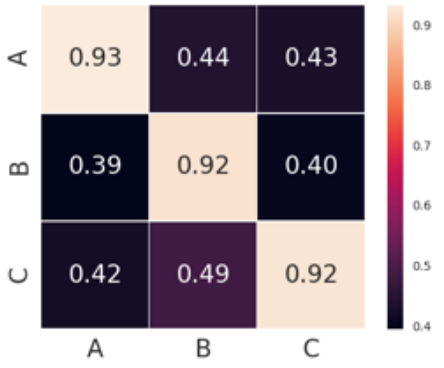
(c) MNIST | DRN-C-26



(d) CIFAR10 | DRN-C-26



(e) MNIST | DLA34



(f) CIFAR10 | DLA34

Figure A.11: Transferability analysis for classification models: cell (i, j) shows the attack success rate of the adversarial examples generated against Model j and evaluate on Model i under targeted attack setting. Model A,B,C are model with the same architecture (DRN-D-22, DRN-C-26 or DLA34UP) and different initialization. All the adversarial examples are generated using fast iterative gradient sign method. The caption of each sub-figure bear the "dataset | model".

APPENDIX B

Appendix of Detecting Adversarial Examples by Using Properties of Data

B.1 Implementation Details

Attack setting As described in the threat model section, we limit the $\mathcal{L}_2$ norm of the adversarial perturbation to be below 8 pixel in all settings because otherwise the the adversarial perturbation would be perceptible and can be easily detected. We set a max iteration of 1000 as the breakout condition.

Adversarial targets of semantic segmentation We evaluate over three different adversarial targets: Remapping, Stripe, and ICCV 2019. "Remapping" means we generate an adversarial target by shifting the numerical label of each class for the prediction from a benign frame by a constant offset. This way, we can guarantee that each target has no overlap with the ground truth. This dynamically generated target also reflects the movement of pixels between frames in a video. For "Stripe", we divide the target into 19 strips evenly, each of which is filled with a class label, aiming to mitigate possible bias for different classes. Finally, "ICCV 2019" places the text "ICCV 2019" over the image with contiguous spaces representing different classes.

Bounding box mIoU We use a metric called bounding box mIoU as the consistency metric for object detection model. Given the target frame $\boldsymbol{X}_t$ and a pseudo frame $\hat{\boldsymbol{X}}_{s\to t}$, we iterate each the bounding box predicted by the detection model from the pseudo frame and compute intersection over union (IoU) value against all the bounding boxes detected in the target frame $\boldsymbol{X}_t$, agnostic of their class labels. The average of the largest IoU value for each bounding box in the pseudo frame is used as our consistency metric. The computation of bounding box mIoU is fully described in Algorithm 6 and the computation of IoU between patches is also illustrated below.

Let a patch be represented by a tuple (x_1, y_1, x_2, y_2) where (x_1, y_1) and (x_2, y_2) are the upper left and lower right corners' coordinates of the patch. Given two patches $P = (x_1, y_1, x_2, y_2), P = (x_1^{'}, y_1^{'}, x_2^{'}, y_2^{'})$, the intersection area of the two patches $\mathcal{A}$ can be computed by $\mathbf{max}(0, \mathbf{min}(x_2, x_2^{'}) - \mathbf{max}(x_1, x_1^{'})) \cdot \mathbf{max}(0, \mathbf{min}(y_2, y_2^{'}) - \mathbf{max}(y_1, y_1^{'}))$. Let w and h denote the width and height of

Algorithm 6: Bounding Box mIoU

input: array of target bounding boxes $\boldsymbol{T}$;
array of predicted bounding boxes $\boldsymbol{P}$;
output: mIoU between two arrays of bounding boxes c;

Initialization : **cs** $\leftarrow$[], $N \leftarrow \boldsymbol{T}.length, M \leftarrow \boldsymbol{P}.length$;
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{IoU} \leftarrow []$;
  /* Iterate over the bounding boxes in $\boldsymbol{P}$. Each bounding box is represented as a patch with its coordinates. */;
  $\mathbf{B} \leftarrow \boldsymbol{P}[i]$;
  for $j \leftarrow 1$ **to** M **do**
    /* Iterate over the bounding boxes in $\boldsymbol{T}$. */;
    $\mathbf{B}' \leftarrow \boldsymbol{T}[j]$;
    /* getIoU is a function that calculate the IoU between two patches. */;
    $\mathbf{IoU} \overset{+}{\leftarrow} \mathbf{getIoU}(\mathbf{B}, \mathbf{B}')$;
  end
  /* Get the IoU of the bounding box in $\mathbf{T}$ with the largest overlap with $\mathbf{B}$ */;
  $\mathbf{cs} \overset{+}{\leftarrow} \mathbf{Max}(\mathbf{IoU})$;
end
$c \leftarrow \mathbf{Mean}(\mathbf{cs})$;
Return: c

P and w', h' denote the width and height of P'. $\mathbf{getIoU}(P, P')$ is defined as

$$\frac{\mathcal{A}}{w \cdot h + w' \cdot h' - \mathcal{A}} \tag{B.1}$$

Consistency measurement function for action recognition In our paper, we leverage the attack method proposed by Wei et al. [120]. The target action recognition framework considered in that work [120] is a CNN+RNN model, so we use the same model to evaluate *AdvIT* . Let $\boldsymbol{X} = \{\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_N\}$ be a video where $\boldsymbol{X}_i$ is the ith frame. Each frame is first fed into a Inception V2 [109] model individually and the extracted features are further processed by a LSTM model sequentially. The LSTM model outputs class scores (logits) $\boldsymbol{Y}_i$ for each frame; the logits of all frames are averaged and the class with the highest score is assigned to $\boldsymbol{X}$. The sparse adversarial perturbations are generated using the same method described in the original work [120]. We consider the activations of all frames in our consistency metric to determine if a whole video clip is adversarial. Let $\hat{\boldsymbol{X}} = \{\hat{\boldsymbol{X}}_1, \hat{\boldsymbol{X}}_2, \ldots, \hat{\boldsymbol{X}}_N\}$ be a sequence of pseudo-frames generated by warping

Task	Attack Method	Target	Previous Frames	Detection (k)		
				1	3	5
Semantic Segmentation	Houdini	ICCV	Benign	100%	100%	100%
			Adversarial	100%	100%	100%
		Remapping	Benign	100%	100%	100%
			Adversarial	100%	100%	100%
		Stripe	Benign	100%	100%	100%
			Adversarial	100%	100%	100%
	DAG	ICCV	Benign	100%	100%	100%
			Adversarial	100%	100%	100%
		Remapping	Benign	100%	100%	100%
			Adversarial	100%	100%	100%
		Stripe	Benign	100%	100%	100%
			Adversarial	100%	100%	100%
Human Pose Estimation	Houdini	shuffle	Benign	100%	100%	100%
			Adversarial	100%	100%	100%
		Transpose	Benign	100%	100%	100%
			Adversarial	98%	99%	100%
Object Detection	DAG	all	Benign	100%	100%	100%
			Adversarial	100%	100%	100%
		person	Benign	99%	100%	100 %
			Adversarial	97%	98%	100%

Table B.1: Detection results (AUC) of *AdvIT* against *independent frame attack* on various video tasks with different attack methods and targets.

with optical flow and $\hat{\boldsymbol{Y}}_i$ be the logit of the ith frame after replacing $\boldsymbol{X}$ with $\hat{\boldsymbol{X}}$. We concatenate all the logits $\boldsymbol{Y}_i$ ($\hat{\boldsymbol{Y}}_i$ for pseudo frames) together to form a one-dimensional vector with $N \cdot C$ elements where C is the class number. Let $\boldsymbol{F}$ and $\hat{\boldsymbol{F}}$ be two one-dimensional vectors by taking the softmax of the concatenations of $\boldsymbol{Y}_i$s and $\hat{\boldsymbol{Y}}_i$s respectively. $\boldsymbol{F}$ and $\hat{\boldsymbol{F}}$ represent two $N \cdot C$-way categorical distributions, $\boldsymbol{P}_F$ and $\boldsymbol{P}_{\hat{F}}$. We take the average of the forward and backward KL divergence between the two distributions, $KL\left(\boldsymbol{P}_F||\boldsymbol{P}_{\hat{F}}\right) + KL\left(\boldsymbol{P}_{\hat{F}}||\boldsymbol{P}_F\right)$, as the consistency metric between $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$.

Adaptive attack algorithm Let's use $\boldsymbol{R}$[1] to denote a flow estimator. It takes two frames $\boldsymbol{X}_{t-i}, \boldsymbol{X}_t$ as input, where $\boldsymbol{X}_t$ is the current frame and $\boldsymbol{X}_{t-i}$ is the ith previous frame, and outputs the flow $O_F = (\boldsymbol{\Delta u}, \boldsymbol{\Delta v})$. We formulate the above as $(\boldsymbol{\Delta u}, \boldsymbol{\Delta v}) = \boldsymbol{R}(\boldsymbol{X}_{t-i}, \boldsymbol{X}_t)$. Denote $\boldsymbol{Y}_t^a$ as the adversarial target for frame t, l as the loss considered by the attack algorithms (e.g. Houdini [27] and DAG [132]) and g as the targeted machine learning model. For the vanilla attack algorithm, they try to generate the adversarial perturbation $\boldsymbol{E}_t$ by optimizing the following objective:

[1] In our paper, we use Flownet [55] which is differentiable.

$$l(g(\boldsymbol{X}_t + \boldsymbol{E}_t), \boldsymbol{Y}_t^a) \tag{B.2}$$

To perform adaptive attack, the attacker incorporates temporal continuity into the attack. It generates a perturbation for the current frame and the perturbation can fool both current frame and the pseudo frames. The adaptive attack objective is defined as follows:

$$l(g(\boldsymbol{X}_t + \boldsymbol{E}_t), \boldsymbol{Y}_t^a) + \sum_{i=1}^{k} l(\mathbf{warp}(\boldsymbol{R}(\boldsymbol{X}_{t-i}, \boldsymbol{X}_t + \boldsymbol{E}_t) + \boldsymbol{\alpha}, \boldsymbol{X}_{t-i}), \boldsymbol{Y}_t^a) \tag{B.3}$$

where k is number of the previous frames considered for adversarial detection. $\mathbf{warp}$ represents the function to generate the pseudo frames by using formulation (1). $\boldsymbol{\alpha}$ is random noise drawn from $N(0, \sigma^2)$ independently. Due to the randomness, we generate adversarial perturbation using Expectation Over Transformation in Athalye et al. [6]. We follow the settings in Athalye et al. [6] and sample N_z $\boldsymbol{\alpha}$s in each iteration.

$$l(g(\boldsymbol{X}_t + \boldsymbol{E}_t), \boldsymbol{Y}_t^a) + 1/N_z \cdot \sum_{z=0}^{N_z} \sum_{i=1}^{k} l(\mathbf{warp}(\boldsymbol{R}(\boldsymbol{X}_{t-i}, \boldsymbol{X}_t + \boldsymbol{E}_t) + \boldsymbol{\alpha}_z, \boldsymbol{X}_{t_i}), \boldsymbol{Y}_t^a) \tag{B.4}$$

We set $N_z = 30$ in our attack.

Task	Attack Method	Previous Frames	Accuracy[RMS](k) 1	3	5
Semantic Segmentation	Houdini	Benign	0.045± 0.006	0.054±0.011	0.059±0.013
		Adversarial	0.045±0.006	0.054±0.011	0.060±0.013
Human Pose Estimation	Houdini	Benign	0.098±0.010	0.114±0.027	0.127±0.039
		Adversarial	0.101±0.010	0.117±0.026	0.130±0.039
Object Detection	DAG	Benign	0.069±0.017	0.090±0.027	0.104±0.032
		Adversarial	0.069±0.017	0.090±0.027	0.104±0.032

Table B.2: Accuracy of the predicted pseudo-frames among different settings

Interval (k)	Segmentation (ϵ) 2	16	32	Object detection (ϵ) 2	16	32	Human pose (ϵ) 2	16	32
1	100%	100%	100%	98.3%	88.3%	83.5%	98.4%	97.9%	96.3%
3	100%	100%	100%	99.8%	95.5%	91.2%	98.6%	98.4%	96.6%
5	100%	100%	100%	99.9%	97.9%	95.15%	98.7%	98.6%	96.7%

Table B.3: Detection results (AUC) of differnt attack strength

B.2 Experimental Results

We include additional results of *AdvIT* in Table B.1. It shows that our method can achieve almost 100% detection rate among all settings.

Table B.2 evaluate the accuracy of pseudo-frame prediction in terms of Root Mean Square(RMS) error. We observe that the prediction accuracy does not significantly affect the adversarial detection rate.

As long as the prediction accuracy is reasonable, the inconsistency phenomenon is very obvious due to adversarial perturbation. We also show experiments results with different randomness (α) added to the optical flow by varying σ. For semantic segmentation, 0.02 and 0.2 are considered for the value of σ. The corresponding RMS values are 0.088$\pm$ 0.003 and 0.16 ± 0.006 respectively. 100% detection rates are achieved in both settings. It shows that even we more randomness to (α) during wrapping to make the pipeline more robust against adaptive attack, the detection efficacy of *AdvIT* is not compromised.

In addition, we conducted extra experiments by limiting the perturbation magnitude to 2, 16, 32 pixels (in range of [0,255]) to evaluate the effectiveness of *AdvIT* against the attack with different strength in Table B.3. It shows that the detection rate will decrease a bit with the magnitude increasing. But with ensemble of previous k frames, it is still effective.

BIBLIOGRAPHY

[1] Davis challenge 2017. https://davischallenge.org/challenge2017.html.

[2] Youtube video. https://www.youtube.com/watch?v=vgusHl1Oue0.

[3] A. Al-Dujaili and U.-M. O'Reilly. There are no bit parts for sign bits in black-box attacks. *arXiv preprint arXiv:1902.06894*, 2019.

[4] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[5] A. Athalye and I. Sutskever. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 80:284–293, 2017.

[6] A. Athalye, N. Carlini, and D. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[7] S. Baluja and I. Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *arXiv preprint arXiv:1703.09387*, 2017.

[8] P. L. Bartlett and M. H. Wegkamp. Classification with a reject option using a hinge loss. *Journal of Machine Learning Research*, 9(Aug):1823–1840, 2008.

[9] A. N. Bhagoji, W. He, B. Li, and D. Song. Exploring the space of black-box attacks on deep neural networks. *arXiv preprint arXiv:1712.09491*, 2017.

[10] B. Bitterli. Rendering resources, 2016. https://benedikt-bitterli.me/resources/.

[11] W. Brendel, J. Rauber, and M. Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.

[12] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla. Segmentation and recognition using structure from motion point clouds. In *ECCV '08 Proceedings of the 10th European Conference on Computer Vision: Part I*, pages 44–57, 2008.

[13] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer. Adversarial patch. *CoRR*, abs/1712.09665, 2017.

[14] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In *arXiv preprint arXiv:1812.08008*, 2018.

[15] N. Carlini and D. Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*, 2016.

[16] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy, 2017*, 2017.

[17] N. Carlini and D. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.

[18] N. Carlini and D. A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57, 2017. doi: 10.1109/SP.2017.49. URL `https://doi.org/10.1109/SP.2017.49`.

[19] T. F. Chan and C.-K. Wong. Total variation blind deconvolution. *IEEE transactions on Image Processing*, 7(3):370–375, 1998.

[20] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[21] J. Chen, M. I. Jordan, and M. J. Wainwright. Hopskipjumpattack: A query-efficient decision-based attack. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1277–1294. IEEE, 2020.

[22] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. ACM, 2017.

[23] W. Chen, H. Wang, Y. Li, H. Su, Z. Wang, C. Tu, D. Lischinski, D. Cohen-Or, and B. Chen. Synthesizing training images for boosting human 3d pose estimation. In *3D Vision (3DV)*, 2015.

[24] M. Cheng, S. Singh, P.-Y. Chen, S. Liu, and C.-J. Hsieh. Sign-opt: A query-efficient hard-label adversarial attack. *arXiv preprint arXiv:1909.10773*, 2019.

[25] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136, 2008.

[26] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems*, pages 2843–2851, 2012.

[27] M. Cisse, Y. Adi, N. Neverova, and J. Keshet. Houdini: Fooling deep structured prediction models. *arXiv preprint arXiv:1707.05373*, 2017.

[28] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[29] W. Cui, Y. Wang, Y. Fan, Y. Feng, and T. Lei. Localized fcm clustering with spatial information for medical image segmentation and bias field estimation. *Journal of Biomedical Imaging*, 2013:13, 2013.

[30] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, L. Chen, M. E. Kounavis, and D. H. Chau. Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression. *arXiv preprint arXiv:1705.02900*, 2017.

[31] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[32] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, June 2009. doi: 10.1109/CVPR.2009.5206848.

[33] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE ICCV*, pages 2758–2766, 2015.

[34] H. Drucker, D. Wu, and V. N. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural networks*, 10(5):1048–1054, 1999.

[35] K. Dvijotham, S. Gowal, R. Stanforth, R. Arandjelovic, B. O'Donoghue, J. Uesato, and P. Kohli. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*, 2018.

[36] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy. A study of the effect of jpg compression on adversarial images. *arXiv preprint arXiv:1608.00853*, 2016.

[37] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan. 2015.

[38] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song. Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945*, 1, 2017.

[39] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, F. Tramer, A. Prakash, T. Kohno, and D. Song. Physical adversarial examples for object detectors. *arXiv preprint arXiv:1807.07769*, 2018.

[40] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[41] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[42] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, T. Mann, and P. Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[43] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*, 2016.

[44] S. Gu and L. Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

[45] C. Guo, M. Rana, M. Cisse, and L. van der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[46] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla. Understanding realworld indoor scenes with synthetic data. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4077–4085, 2016.

[47] B. G. Haskell and A. Puri. *MPEG Video Compression Basics*, pages 7–38. Springer New York, New York, NY, 2012. ISBN 978-1-4419-6184-6. doi: 10.1007/978-1-4419-6184-6_2. URL https://doi.org/10.1007/978-1-4419-6184-6_2.

[48] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[49] W. He, J. Wei, X. Chen, N. Carlini, and D. Song. Adversarial example defense: Ensembles of weak defenses are not strong. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, 2017. USENIX Association. URL https://www.usenix.org/conference/woot17/workshop-program/presentation/he.

[50] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[51] Y. Hold-Geoffroy, K. Sunkavalli, S. Hadap, E. Gambaretto, and J.-F. Lalonde. Deep outdoor illumination estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2017.

[52] L. Hosek and A. Wilkie. An analytic model for full spectral sky-dome radiance. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2012)*, 31(4), July 2012. To appear.

[53] H. Hosseini, Y. Chen, S. Kannan, B. Zhang, and R. Poovendran. Blocking transferability of adversarial examples in black-box learning systems. *arXiv preprint arXiv:1703.04318*, 2017.

[54] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[55] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE conference on CVPR*, volume 2, page 6, 2017.

[56] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin. Black-box adversarial attacks with limited queries and information. *arXiv preprint arXiv:1804.08598*, 2018.

[57] D. S. Immel, M. F. Cohen, and D. P. Greenberg. A radiosity method for non-diffuse environments. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 133–142, New York, NY, USA, 1986. ACM. ISBN 0-89791-196-2. doi: 10.1145/15922.15901. URL http://doi.acm.org/10.1145/15922.15901.

[58] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[59] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *NIPS*, pages 2017–2025, 2015.

[60] W. Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[61] Y. Jang, T. Zhao, S. Hong, and H. Lee. Adversarial defense via learning to generate diverse attacks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2740–2749, 2019.

[62] B. Johnson and Z. Xie. Unsupervised image segmentation evaluation and refinement using a multi-scale approach. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(4): 473–483, 2011.

[63] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.

[64] J. T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM. ISBN 0-89791-196-2. doi: 10.1145/15922.15902. URL http://doi.acm.org/10.1145/15922.15902.

[65] S. Kandaswamy, K. Forbus, and D. Gentner. Modeling learning via progressive alignment using interim generalizations. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 36, 2014.

[66] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[67] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[68] N. Kolotouros. Pytorch implememtation of the neural mesh renderer. https://github.com/daniilidis-group/neural_renderer, 2018. Accessed: 2018-09-10.

[69] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pages 109–117, 2011.

[70] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[71] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[72] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[73] Y. LeCun and C. Cortes. The MNIST database of handwritten digits. 1998.

[74] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 17(39):1–40, 2016.

[75] Q. Li, L. Chen, M. Li, S.-L. Shaw, and A. Nüchter. A sensor-fusion drivable-region and lane-detection system for autonomous vehicle navigation in challenging road scenarios. *IEEE Transactions on Vehicular Technology*, 63(2):540–555, 2013.

[76] G. Lin, C. Shen, A. Van Den Hengel, and I. Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3194–3203, 2016.

[77] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[78] C. Liu et al. *Beyond pixels: exploring new representations and applications for motion analysis*. PhD thesis, Massachusetts Institute of Technology, 2009.

[79] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[80] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

[81] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[82] J. Lu, H. Sibai, E. Fabry, and D. Forsyth. No need to worry about adversarial examples in object detection in autonomous vehicles. *arXiv preprint arXiv:1707.03501*, 2017.

[83] Y. Luo, X. Boix, G. Roig, T. Poggio, and Q. Zhao. Foveation-based mechanisms alleviate adversarial examples. *arXiv preprint arXiv:1511.06292*, 2015.

[84] X. Ma, B. Li, Y. Wang, S. M. Erfani, S. Wijewickrema, M. E. Houle, G. Schoenebeck, D. Song, and J. Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*, 2018.

[85] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[86] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. *arXiv preprint ArXiv:1611.04076*, 2016.

[87] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt and Co., Inc., New York, NY, USA, 1982. ISBN 0716715678.

[88] F. Massa, B. Russell, and M. Aubry. Deep exemplar 2d-3d detection by adapting from real to rendered views. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[89] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison. Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation? In *The IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[90] M. Mirman, T. Gehr, and M. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3575–3583, 2018.

[91] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[92] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1765–1773, 2017.

[93] A. Mdry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv:1706.06083 [cs, stat]*, June 2017.

[94] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*, pages 483–499. Springer, 2016.

[95] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[96] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[97] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[98] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[99] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: Ground truth from computer games. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016.

[100] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.

[101] P. K. Saha, J. K. Udupa, and D. Odhner. Scale-based fuzzy connected image segmentation: theory, algorithms, and validation. *Computer Vision and Image Understanding*, 77(2): 145–174, 2000.

[102] A. Shafahi, M. Najibi, M. A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor, and T. Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pages 3358–3369, 2019.

[103] C. E. Shannon. Communication theory of secrecy systems. *Bell Labs Technical Journal*, 28 (4):656–715, 1949.

[104] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic scene completion from a single depth image. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[105] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[106] H. Su, C. R. Qi, Y. Li, and L. J. Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *The IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[107] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[108] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[109] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[110] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[111] P. Tabacof and E. Valle. Exploring the space of adversarial images. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 426–433. IEEE, 2016.

[112] F. Tramèr, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[113] F. Tramèr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[114] C.-C. Tu, P. Ting, P.-Y. Chen, S. Liu, H. Zhang, J. Yi, C.-J. Hsieh, and S.-M. Cheng. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 742–749, 2019.

[115] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 311–318, New York, NY, USA, 1994. ACM. ISBN 0-89791-667-0. doi: 10.1145/192161.192241. URL http://doi.acm.org/10.1145/192161.192241.

[116] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid. Learning from synthetic humans. In *CVPR*, 2017.

[117] A. J. Viterbi. An intuitive justification and a simplified implementation of the map decoder for convolutional codes. *IEEE Journal on Selected Areas in Communications*, 16(2):260–264, 1998.

[118] C. R. Vogel and M. E. Oman. Iterative methods for total variation denoising. *SIAM Journal on Scientific Computing*, 17(1):227–238, 1996.

[119] S. Wang, Y. Chen, A. Abdou, and S. Jana. Mixtrain: Scalable training of formally robust neural networks. *arXiv preprint arXiv:1811.02625*, 2018.

[120] X. Wei, J. Zhu, and H. Su. Sparse adversarial perturbations for videos. *arXiv preprint arXiv:1803.02536*, 2018.

[121] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.

[122] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BkUHlMZ0b.

[123] E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5283–5292, 2018.

[124] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[125] C. Xiang, C. R. Qi, and B. Li. Generating 3d adversarial point clouds. *arXiv preprint arXiv:1809.07016*, 2018.

[126] Y. Xiang, R. Mottaghi, and S. Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *Applications of Computer Vision (WACV), 2014 IEEE Winter Conference on*, pages 75–82. IEEE, 2014.

[127] C. Xiao, R. Deng, B. Li, F. Yu, D. Song, et al. Characterizing adversarial examples based on spatial consistency information for semantic segmentation. In *Proceedings of the (ECCV)*, pages 217–234, 2018.

[128] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song. Generating adversarial examples with adversarial networks. *arXiv preprint arXiv:1801.02610*, 2018.

[129] C. Xiao, D. Yang, B. Li, J. Deng, and M. Liu. Meshadv: Adversarial meshes for visual recognition. In *CVPR*, 2018.

[130] C. Xiao, J.-Y. Zhu, B. Li, W. He, M. Liu, and D. Song. Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*, 2018.

[131] C. Xiao, R. Deng, B. Li, T. Lee, B. Edwards, J. Yi, D. Song, M. Liu, and I. Molloy. Advit: Adversarial frames identifier based on temporal consistency in videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3968–3977, 2019.

[132] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille. Adversarial examples for semantic segmentation and object detection. In *International Conference on Computer Vision*. IEEE, 2017.

[133] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018.

[134] C. Xie, M. Tan, B. Gong, J. Wang, A. Yuille, and Q. V. Le. Adversarial examples improve image recognition. *arXiv preprint arXiv:1911.09665*, 2019.

[135] H. Xu, Y. Gao, F. Yu, and T. Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182, 2017.

[136] D. Yang and J. Deng. Shape from shading through shape evolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[137] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, 2016.

[138] F. Yu, V. Koltun, and T. Funkhouser. Dilated residual networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[139] F. Yu, D. Wang, and T. Darrell. Deep layer aggregation. *arXiv preprint arXiv:1707.06484*, 2017.

[140] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

[141] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[142] X. Zeng, C. Liu, W. Qiu, L. Xie, Y.-W. Tai, C. K. Tang, and A. L. Yuille. Adversarial attacks beyond the image space. *arXiv preprint arXiv:1711.07183*, 2017.

[143] D. Zhang, T. Zhang, Y. Lu, Z. Zhu, and B. Dong. You only propagate once: Painless adversarial training using maximal principle. *arXiv preprint arXiv:1905.00877*, 2(3), 2019.

[144] H. Zhang and J. Wang. Joint adversarial training: Incorporating both spatial and pixel attacks. *arXiv preprint arXiv:1907.10737*, 2019.

[145] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.

[146] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016.

[147] Y. Zhang, S. Song, E. Yumer, M. Savva, J.-Y. Lee, H. Jin, and T. Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[148] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890, 2017.

[149] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros. View synthesis by appearance flow. In *ECCV*, pages 286–301. Springer, 2016.

[150] C. Zhu, Y. Cheng, Z. Gan, S. Sun, T. Goldstein, and J. Liu. Freelb: Enhanced adversarial training for language understanding. *arXiv preprint arXiv:1909.11764*, 2019.

[151] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *ICCV*, 2017.