Towards Designing Robust Deep Learning Models for 3D Understanding

Dissertation by
Abdullah Hamdi

In Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

King Abdullah University of Science and Technology (KAUST)
Thuwal, Kingdom of Saudi Arabia

©April, 2023
Abdullah Hamdi
All rights reserved
https://orcid.org/0000-0003-3989-7540
ABSTRACT

Towards Designing Robust Deep Learning models for 3D Understanding
Abdullah Hamdi

This dissertation presents novel methods for addressing important challenges related to the robustness of Deep Neural Networks (DNNs) for 3D understanding and in 3D setups. Our research focuses on two main areas, adversarial robustness on 3D data and setups and the robustness of DNNs to realistic 3D scenarios. One paradigm for 3D understanding is to represent 3D as a set of 3D points and learn functions on this set directly. Our first work, AdvPC, addresses the issue of limited transferability and ease of defense against current 3D point cloud adversarial attacks. By using a point cloud Auto-Encoder to generate more transferable attacks, AdvPC surpasses state-of-the-art attacks by a large margin on 3D point cloud attack transferability. Additionally, AdvPC increases the ability to break defenses by up to 38% as compared to other baseline attacks on the ModelNet40 dataset. Another paradigm of 3D understanding is to perform 2D processing of multiple images of the 3D data. The second work, MVTN, addresses the problem of selecting viewpoints for 3D shape recognition using a Multi-View Transformation Network (MVTN) to learn optimal viewpoints. It combines MVTN with multi-view approaches leading to state-of-the-art results on standard benchmarks ModelNet40, ShapeNet Core55, and ScanObjectNN. MVTN also improves robustness to realistic scenarios like rotation and occlusion.

Our third work analyzes the Semantic Robustness of 2D Deep Neural Networks, addressing the problem of high sensitivity toward semantic primitives in DNNs by visualizing the DNN global behavior as semantic maps and observing the interesting behavior of some DNNs. Additionally, we develop a bottom-up approach to detect
robust regions of DNNs for scalable semantic robustness analysis and benchmarking of different DNNs. The fourth work, **SADA**, showcases the problem of lack of robustness in DNNs specifically for the safety-critical applications of autonomous navigation, beyond the simple classification setup. We present a general framework (BBGAN) for black-box adversarial attacks on trained agents, which covers semantic perturbations to the environment of the agent performing the task. BBGAN is trained to generate failure cases that consistently fool a trained agent on tasks such as object detection, self-driving, and autonomous UAV racing.
I extend my sincerest gratitude to my advisor, Bernard Ghanem, for his unwavering support and guidance throughout my academic journey. He not only nurtured my growth as a researcher but also as a person, instilling in me a passion for academic excellence and a desire to tackle challenging computer vision and machine learning problems. He has been more than just an advisor, but also a friend and role model in both my professional and personal life. I am eternally grateful for the support and guidance he has provided, and will always be thankful for the impact he has had on my life.

I would like also to thank Prof. Matthias Niessner for his support and mentorship during my stay at TUM for an internship. His honesty and hard work opened my eyes to a different reality than the one I am accustomed to. This broadened my perspective of what it means to do great work with a unique perspective.

I would like to acknowledge and thank my collaborators, Matthias Muller, Adel Bibi, Silvio Ginancola, Modar Alfadly, Ali Thabet, and Sara Rojas Martinez for their contributions to my research and for being wonderful friends. I would also like to thank the members of the IVUL group for their support, camaraderie, and memories.

Lastly, I would like to express my deepest love and gratitude to my parents, Jamal and Wafaa, and my beloved wife, Sarah, for their unwavering support and guidance. They are the pillars of my strength and the source of any beauty in my life. This dissertation is dedicated to them.
The dissertation of Abdullah Hamdi is approved by the examination committee

Committee Chairperson: Prof. Bernard Ghanem
Committee Members: Prof. Mohamed Elhoseiny, Prof. Shinkyu Park, Prof. Jia-Bin Haung
TABLE OF CONTENTS

Abstract ........................................... 2
Acknowledgements .................................. 4
Examination Committee Page ....................... 5
List of Figures ...................................... 12
List of Tables ....................................... 20

1 Introduction ...................................... 25
  1.1 Motivation .................................... 25
  1.2 Robustness of 3D Models ...................... 26
    1.2.1 3D point Clouds ......................... 26
    1.2.2 Multi-view 2D images .................... 26
  1.3 Physical Robustness of 2D Models ............ 27
    1.3.1 Analysing 2D DNN Classifiers in 3D Setups .. 27
    1.3.2 Robustness of Vision-based Autonomous Applications .. 27
  1.4 Presentation and Structure .................. 28

2 AdvPC: Transferable Adversarial Perturbations on 3D Point Clouds  30
  2.1 Related Work ................................ 31
    2.1.1 Deep Learning for 3D Point Clouds ........ 31
    2.1.2 Adversarial Attacks ..................... 32
  2.2 Methodology ................................ 34
    2.2.1 AdvPC Attack Pipeline ................... 34
    2.2.2 AdvPC Attack Loss ....................... 36
  2.3 Experiments .................................. 38
    2.3.1 Setup .................................... 38
    2.3.2 Results .................................. 41
  2.4 Analysis ..................................... 43
## 3 MVTN: Multi-View Transformation Network for 3D Shape Recognition

3.1 Related Work .............................................. 46
3.2 Methodology .............................................. 47
  3.2.1 Overview of Multi-View 3D Recognition ................. 47
  3.2.2 Multi-View Transformation Network (MVTN) ............ 49
3.3 Experiments .............................................. 52
  3.3.1 Datasets ............................................ 52
  3.3.2 Metrics ............................................ 52
  3.3.3 Baselines .......................................... 53
  3.3.4 MVTN Details ....................................... 53
3.4 Results .................................................. 56
  3.4.1 3D Shape Classification ............................... 57
  3.4.2 3D Shape Retrieval ................................... 58
  3.4.3 Rotation Robustness ................................... 59
  3.4.4 Occlusion Robustness .................................. 59
3.5 Analysis and Insights .................................... 61
  3.5.1 Ablation Study ....................................... 61
  3.5.2 Time and Memory Requirements ......................... 63

## 4 Towards Analyzing Semantic Robustness of Deep Neural Networks

4.1 Related Work .............................................. 66
  4.1.1 Understanding Deep Neural Networks .................... 66
  4.1.2 Semantic Adversarial Attacks on Deep Neural Networks .. 67
  4.1.3 Optimizing Integral Bounds ............................. 67
4.2 Methodology .............................................. 69
  4.2.1 Region Finding as an Operator ........................ 70
  4.2.2 Deriving Update Directions ............................. 72
4.3 Experiments .............................................. 76
  4.3.1 Setup and Data ....................................... 76
  4.3.2 Mapping the Networks .................................. 77
  4.3.3 Growing Semantic Robust Regions ....................... 77
  4.3.4 Applications ......................................... 77
4.4 Analysis .................................................. 80
## SADA: Semantic Adversarial Diagnostic Attacks for Autonomous Applications

5.1 Related Work .................................................. 82
  5.1.1 Semantic Attacks and Simulations ..................... 82
  5.1.2 Adversarial Attacks and Reinforcement Learning ...... 83
5.2 Methodology ................................................. 84
  5.2.1 Generalizing Adversarial Attacks .................... 84
  5.2.2 Black-Box Generative Adversarial Network ............ 86
  5.2.3 Special Cases of Adversarial Attacks ................. 88
5.3 Applications ................................................ 89
  5.3.1 Object Detection ....................................... 89
  5.3.2 Self-Driving .......................................... 90
  5.3.3 UAV Racing ........................................... 90
5.4 Experiments ................................................ 92
  5.4.1 BBGAN ................................................. 92
  5.4.2 Testing, Evaluation, and Baselines ................... 93
  5.4.3 Results ............................................... 94
  5.4.4 Analysis .............................................. 95
5.5 Insights from SADA ........................................ 96

## Conclusion and Future Work

6.1 Summary .................................................. 98
6.2 Future Work ................................................ 99

References ......................................................... 100

Appendices ......................................................... 116

  A.1 Background on Point Cloud Distances .................... 117
    A.1.1 Trivial Distances ($\ell_p$) .......................... 117
    A.1.2 Non-trivial Distances ................................ 118
  A.2 Full Transferability Results ............................ 119
    A.2.1 Transferability on Different Norms ................. 119
  A.3 Defenses Results ......................................... 119
  A.4 Analysis of the results .................................. 119
    A.4.1 Ablation Study (hyperparameter $\gamma$) ........... 119
    A.4.2 Network Sensitivity to Point Cloud Attacks ....... 123
    A.4.3 Effect of the Auto-Encoder (AE) .................... 124
A.4.4 Ablation Study on the Losses ........................................ 124
B.1 Detailed Experimental Setup ........................................... 127
  B.1.1 Datasets ................................................................. 127
  B.1.2 MVTN Details .......................................................... 128
B.2 Additional Results ......................................................... 134
  B.2.1 Classification and Retrieval Benchmarks ......................... 134
  B.2.2 Rotation Robustness ................................................. 134
  B.2.3 Occlusion Robustness ................................................. 135
B.3 Analysis and Insights ..................................................... 140
  B.3.1 Ablation Study ......................................................... 140
  B.3.2 Transferability of MVTN View-Points ............................. 142
  B.3.3 Shape Retrieval Examples ........................................... 143
C.1 Analyzing Deep Neural Networks ......................................... 146
  C.1.1 Networks Semantic Maps (1D) .................................... 146
  C.1.2 Networks Semantic Maps (2D) .................................... 146
  C.1.3 Examples of Found Regions ( with Example Renderings) .... 146
  C.1.4 Analyzing Semantic Data Bias in ImageNet .................... 147
C.2 Detailed Derivations of the Update Directions of the Bounds .... 147
  C.2.1 Defining Robustness as an Operator ............................. 147
  C.2.2 Divergence of the Bounds ......................................... 148
  C.2.3 Naive Approach ...................................................... 149
  C.2.4 Outer- Inner Ratio Loss (OIR) ................................... 152
  C.2.5 Trapezoidal Approximation Formulation ....................... 161
C.3 Analysis ................................................................. 164
  C.3.1 Detected Robust Regions .......................................... 164
  C.3.2 Hyper-Parameters .................................................. 165
  C.3.3 Dataset ............................................................... 165
D.1 Empirical Justification for BBGAN ..................................... 171
  D.1.1 Empirical Proof .................................................... 171
D.2 Special Cases of Generic Adversarial attacks: ..................... 173
  D.2.1 Pixel Adversarial Attack on Image Classifiers ................. 173
  D.2.2 Semantic Adversarial Attack on Object Detectors ............. 175
  D.2.3 Semantic Adversarial Attack on Autonomous Agents .......... 176
D.3 Boosting Strategy for BBGAN .......................................... 177
  D.3.1 Intuition for Boosting ............................................. 177
  D.3.2 Description of Boosting for BBGANs ........................... 177
LIST OF FIGURES

2.1 Transferable Adversarial Perturbations on 3D point clouds: Generating adversarial attacks to fool PointNet [1](PN) by perturbing a Table point cloud. The perturbed 3D object not only forces PointNet to predict an incorrect class, but also induces misclassification on other unseen 3D networks (PointNet++ [2], DGCNN [3]) that are not involved in generating the perturbation. Fooling unseen networks poses a threat to 3D deep vision models. ................................. 31

2.2 AdvPC Attack Pipeline: We optimize for the constrained perturbation variable $\Delta$ to generate the perturbed sample $\mathcal{X}' = \mathcal{X} + \Delta$. The perturbed sample fools a trained classifier $\mathbf{F}$ (i.e. $\mathbf{F}(\mathcal{X}')$ is incorrect), and at the same time, if the perturbed sample is reconstructed by an Auto-Encoder (AE) $\mathbf{G}$, it too fools the classifier (i.e. $\mathbf{F}(\mathbf{G}(\mathcal{X}'))$ is incorrect). The AdvPC loss for network $\mathbf{F}$ is defined in Eq (2.6) and has two parts: network adversarial loss (purple) and data adversarial loss (green). ................................. 34

2.3 Examples of Targeted AdvPC Attacks: Adversarial attacks are generated for victim networks PointNet, PointNet ++ (MSG/SSG) and DGCNN using AdvPC. The unperturbed point clouds are in black (top) while the perturbed examples are in blue (bottom). The network predictions are shown under each point cloud. ................................. 37

2.4 Transferability Across Different Norm-Budgets: Here, the victim network is DGCNN [3] and the attacks are optimized using different $\epsilon_\infty$ norm-budgets. We report the attack success on DGCNN and on the transfer networks (PointNet, PointNet ++ MSG, and PointNet++ SSG). We note that our AdvPC transfers better to the other networks across different $\epsilon_\infty$ as compared to the baselines 3D-Adv[4] and KNN Attack [5].. ................................. 40
2.5 **Transferability Matrix**: Visualizing the overall transferability for 3D-Adv [4] (*left*), KNN Attack [5] (*middle*), and our AdvPC (*right*). Elements in the same row correspond to the same victim network used in the attack, while those in the same column correspond to the network that the attack is transferred to. Each matrix element measures the average success rate over the range of $\epsilon_{\infty}$ for the transfer network. Brighter off-diagonal matrix elements indicate better transferability. The transferability score under each matrix is the average of the off-diagonal matrix values, which summarizes overall transferability for an attack.

3.1 **Multi-View Transformation Network (MVTN)**. We propose a differentiable module that predicts the best view-points for a task-specific multi-view network. MVTN is trained jointly with this network without any extra training supervision while improving the performance of 3D classification and shape retrieval.

3.2 **End-to-End Learning Pipeline for Multi-View Recognition**. To learn adaptive scene parameters $u$ that maximize the performance of a multi-view network $C$ for every 3D object shape $S$, we use a differentiable renderer $R$. MVTN extracts coarse features from $S$ by a point encoder and regresses the adaptive scene parameters for that object. In this example, the parameters $u$ are the azimuth and elevation angles of cameras pointing towards the center of the object. The MVTN pipeline is optimized end-to-end for the task loss.

3.3 **Multi-View Camera Configurations**: The view setups commonly used in the multi-view literature are circular [6] or spherical [7, 8]. Our MVTN learns to predict specific view-points for each object shape at inference time. The shape’s center is shown as a red dot, and the view-points as blue cameras with their mesh renderings shown at the bottom.

3.4 **Multi-View Point Cloud Renderings**. We show some examples of point cloud renderings used in our pipeline. Note how point cloud renderings offer more information about content hidden from the camera view-point (*e.g.* car wheels from the occluded side), which can be useful for recognition.

3.5 **Qualitative Examples for Object Retrieval**: (*left*): we show some query objects from the test set. (*right*): we show top five retrieved objects by our MVTN from the training set. Images of negative retrieved objects are framed.
3.6 **Occlusion of 3D Objects**: We simulate realistic occlusion scenarios in 3D point clouds by cropping a percentage of the object along canonical directions. Here, we show an object occluded with different ratios and from different directions.

3.7 **Effect of the Number of Views.** We plot the test accuracy vs. the number of views (M) used to train MVCNN on fixed, random, and learned MVTN view configurations. We observe a consistent 2% improvement with MVTN over a variety of views.

4.1 **Semantic Robustness of Deep Networks.** Trained neural networks can perform poorly when subject to small perturbations in the semantics of the image. *(left)*: We show how perturbing the azimuth viewing angle of a simple *teapot* object can dramatically affect the score of a pretrained InceptionV3 [9] for the *teapot* class. *(right)*: We plot the softmax confidence scores of different DNNs on the same *teapot* object viewed from 360 degrees around the object. For comparison, lab researchers identified the object from all angles.

4.2 **Analysis Pipeline**: We leverage neural mesh renderer R [10] to render shape $S_z$ of class $z$ according to semantic scene parameters $u$. The resulting image is passed to trained network $C$ that is able to identify the class $z$. The behaviour of the softmax score at label $z$ (dubbed $f_z(u)$) is analyzed for different parameters $u$ and for the specific shape $S_z$. In our experiments, we pick $u_1$ and $u_2$ to be the azimuth and elevation angles, respectively.

4.3 **Semantic Robust Region Finding**: We find robust regions of semantic parameters for Rsnet50 [11] and for a *bathtub* object by the three bottom-up formulations (naive, OIR_W, and OIR_B). *(left)*: Semantic space is 1D (azimuth angle of camera) with three initial points. *(right)*: Semantic space is 2D (azimuth angle and elevation angle of camera) with four initial points.

4.4 **Network Semantic Maps**: We plot the 2D semantic maps of four different networks on two shapes of a *chair* class *(top)* and a *cup* class *(bottom)*. InceptionV3 is very confident about its decision, but at the cost of creating semantic “traps”, where sharp performance degradation happens in the middle of a robust region. Color maps follow Figure 4.3 *(right)* color map.

4.5 **Semantic Bias in ImageNet.** By taking the average semantic maps over 10 shapes of *cup* class and over different networks, we can visualize the bias of the training data. The angles of low score are probably not well-represented in ImageNet [12].
5.1 **Semantic Adversarial Diagnostic Attacks.** Neural networks can perform poorly or downright fail when encountering some environments. To diagnose why they fail and how they can be improved, we seek to learn the underlying distribution of semantic parameters, which generate environments that pose difficulty to these networks when applied to three safety critical tasks.

5.2 **Generic Adversarial Attacks on Agents.** \( E_u \) is a parametric environment with which an agent \( A \) interacts. The agent receives an observation \( o_t \) from the environment and produces an action \( a_t \). The environment scores the agent and updates its state until the episode finishes. A final score \( Q(A, E_u) \) is given to the adversary \( G \), which in turn updates itself to propose more adversarial parameters \( u \) for the next episode.

5.3 **BBGAN: Learning Fooling Distribution of Semantic Environment Parameters.** We learn an adversary \( G \), which samples semantic parameters \( u \) that parametrize the environment \( E_u \), such that an agent \( A \) fails in a given task in \( E_u \). The inducer produces the induced set \( S_u' \) from a uniformly sampled set \( \Omega \) by filtering the lowest scoring \( u \) (according to \( Q \) value), and passing \( S_u' \) for BBGAN training. Note that \( Q_1 \leq Q_s \ldots \leq Q_N \), where \( s = |S_u'| \), \( N = |\Omega| \). The inducer and the discriminator are only used during training (dashed lines), after which the adversary learns the fooling distribution \( P_u' \). Three safety-critical applications are used to demonstrate this in three virtual environments: object detection (in Blender [13]), self-driving cars (in CARLA [14]), and autonomous racing UAVs (in Sim4CV [15]).

5.4 **Object Detection Attack Setup:** (Left): the 100 shapes from Pascal3D and ShapeNet of 12 object classes, used to uncover the failure cases of the YOLOV3 detector. (Right): the semantic parameters \( u \) defining the environment.

5.5 **Qualitative Examples:** (Top): BBGAN generated samples that fool YOLOV3 detector on the Truck class. (Bottom): BBGAN generated tracks that fool the UAV navigation agent.

5.6 **Visualization of the Fooling Distribution.** (right): We plot the camera positions and light source directions of 250 sampled parameters in a 3D sphere around the object. (left): We show how real photos of a toy car, captured from the same angles as rendered images, confuse the YOLOV3 detector in the same way.
5.7 Nearest Neighbor in Training: (top): generated fooling samples by our BBGAN for 4 different classes. (bottom): the corresponding NN from the training set. We can see that our BBGAN generates novel fooling parameters that are not present in training.

A.1 Transferability Across Different $\epsilon_\infty$ Norm-Budgets: Here, the attacks are optimized using different $\epsilon_\infty$ norm-budgets. We report the attack success on all victim networks and the success of these attacks on each transfer network.

A.2 Ablation Study in $\ell_\infty$: Studying the effect of changing AdvPC hyperparameter ($\gamma$) on the success rate of the attack (left) and on its transferability (right). The transferability score reported for each victim network is the average success rate on the transfer networks averaged across all different norm-budgets $\epsilon_\infty$.

A.3 Sensitivity of Architectures in $\ell_\infty$: We evaluate the sensitivity of each of the four networks for increasing norm-budget. For each network, we plot the classification accuracy under three attacks. Overall, DGCNN [3] is affected the least by adversarial perturbation.

A.4 Effect of the Auto-Encoder (AE): The AE does not affect the unperturbed point cloud (classified correctly by PN before and after AE). The AE cleans the perturbed point cloud by baseline attacks [4, 5]. However, our AdvPC attack can fool PN before and after AE reconstruction.

B.1 Training Data with Randomized Color and Lighting. We show examples of mesh renderings of ModelNet40 used in training our MVTN. The color of the object and the light’s direction are randomized during training for augmentation purposes and fixed in testing for stable performance. For this figure, eight circular views are shown for each 3D shape.

B.2 ShapeNet Core55. We show some examples of point cloud renderings of ShapeNet Core55 [16] used in training MVTN. Note how point cloud renderings offer more information about content hidden from the camera view-point (e.g. car wheels from the occluded side), which can be useful for recognition. For this figure, 12 spherical views are shown for each 3D shape.
B.3 **ScanObjectNN Variants.** We show examples of point cloud renderings of different variants of the ScanObjectNN [17] point cloud dataset used to train MVTN. The variants are: object only, object with background, and the hardest perturbed variant (with rotation and translation). For this figure, six circular views are shown for each 3D shape.

B.4 **MVTN Network Architecture.** We show a schematic and a code snippet for MVTN-spherical/MVTN-circular regression architectures used, where $b$ is the size of the point features extracted by the point encoder of MVTN and $M$ is the number of views learned. In most of our experiments, $b = 40$, while the output is the azimuth and elevation angles for all the $M$ views used. The network is drawn using [18].

B.5 **Views Configurations.** We show some possible view configurations that can be used with a varying number of views. (a): circular, (b): spherical, (c): random.

B.6 **Robustness on a Varying Y-Rotation.** We study the effect of varying the maximum rotation perturbation on the classification accuracies on ModelNet40. We compare the performance of circular MVCNN [6] to our circular-MVTN when it equips MVCNN when the number of views is 6 and 12. Note how MVTN stabilizes the drop in performance for larger Y-rotation perturbations, and the improvement is more significant for the smaller number of views $M$.

B.7 **Occlusion Robustness of 3D Methods.** We plot test accuracy vs. the Occlusion Ratio of the data to simulate the occlusion robustness of different 3D methods: PointNet [1], DGCNN [3], and MVTN. Our MVTN achieves close to 13% better than PointNet when half of the object is occluded. MVTN$^1$ refers to MVTN with MVCNN as the multi-view network while MVTN$^2$ refers to MVTN with View-GCN as the multi-view network.

B.8 **Occlusion of 3D Objects.** We simulate realistic occlusion scenarios in 3D point clouds by cropping a percentage of the object along canonical directions. Here, we show an object occluded with different ratios and from different directions.
B.9 **Variants of MVTN.** We plot test accuracy vs. the number of views used in training different variants of our MVTN. Note how MVTN-spherical is generally more stable in achieving better performance on ModelNet40. 95% confidence interval is also plotted on each setup (repeated four times).

B.10 **Effect of Distance to 3D Object.** We study the effect of changing the distance on training a spherical ViewGCN. We show that the distance of 2.2 units to the center is in between far and close it and gives the best accuracy.

B.11 **Qualitative Examples for Object Retrieval:** (left): we show some query objects from the test set. (right): we show top ten retrieved objects by our MVTN from the training set.

C.1 **1D Network Semantic Maps NSM:** visualizing 1D Semantic Robustness profile for different networks averaged over 10 different shapes.

C.2 **2D Network Semantic Maps NSM:** visualizing 2D Semantic Robustness profile for different networks averaged over 10 different shapes. Every row is a different class.

C.3 **Qualitative Examples of Robust Regions:** visualizing different runs of the algorithm to find robust regions along with different renderings from inside these regions for those specific shapes used in the experiments.

C.4 **Data Semantic Maps DSM:** visualizing Semantic Data Bias in the common training dataset (i.e. ImageNet [12]) By averaging the Networks Semantic Maps (NSM) of different networks and on different shapes, Different classes have a different semantic bias in ImageNet as clearly shown in the maps above. The symmetry in the maps is attributed to the 3D symmetry of the objects.

D.1 **BBGAN Qualitative Examples in Object Detection** - Some sample images for each class that were rendered according to parameters generated by BBGAN which fooled the object detector.

D.2 **Training Data for YOLOV3 Experiment** - Some sample images from the dataset used for object detection with YOLOV3. Note that in the actual dataset, each object has a random color regardless of its class. For clarity, we uniformly color each class in this figure.

D.3 **Environment for Self-driving** - Some samples of the CARLA [14] simulator environment.
D.4 **Environment for UAV Racing** - Some samples of the Sim4CV [15] simulator environment. .................................................. 188

D.5 **Training Data for 5-Anchors UAV Racing** - Some sample tracks from the dataset with 5 control points. ........................................... 189
## LIST OF TABLES

2.1 **Transferability of Attacks**: We use norm-budgets (max $\ell_\infty$ norm allowed in the perturbation) of $\epsilon_\infty = 0.18$ and $\epsilon_\infty = 0.45$. All the reported results are the untargeted Attack Success Rate (higher numbers are better attacks). **Bold** numbers indicate the most transferable attacks. Our attack consistently achieves better transferability than the other attacks. For reference, the classification accuracies on unperturbed samples for networks PN, PN++(MSG), PN++(SSG) and DGCNN are 92.8%, 91.5%, 91.5%, and 93.7%, respectively.  

2.2 **Attacking Point Cloud Defenses**: We evaluate untargeted attacks using norm-budgets of $\epsilon_\infty = 0.18$ and $\epsilon_\infty = 0.45$ with DGCNN [3] as the victim network under different defenses for 3D point clouds. Similar to before, we report attack success rates (**higher** indicates better attack). AdvPC consistently outperforms the other attacks [4, 5] for all defenses. Note that both the attacks and evaluations are performed on DGCNN, which has an accuracy of 93.7% without input perturbations (for reference).

3.1 **3D Shape Classification on ModelNet40**. We compare MVTN against other methods in 3D classification on ModelNet40 [19]. * indicates results from our rendering setup (differentiable pipeline), while other multi-view results are reported from pre-rendered views. **Bold** denotes the best result in its setup.

3.2 **3D Point Cloud Classification on ScanObjectNN**. We compare the performance of MVTN in 3D point cloud classification on three different variants of ScanObjectNN [17]. The variants include object with background, object only, and the hardest variant.

3.3 **3D Shape Retrieval**. We benchmark the shape retrieval mAP of MVTN on ModelNet40 [19] and ShapeNet Core55 [16, 20]. MVTN achieves the best retrieval performance among recent state-of-the-art methods on both datasets with only 12 views.
3.4 **Rotation Robustness on ModelNet40.** At test time, we randomly rotate objects in ModelNet40 around the Y-axis (gravity) with different ranges and report the overall accuracy. MVTN displays strong robustness to such Y-rotations. 58

3.5 **Occlusion Robustness of 3D Methods.** We report the test accuracy on point cloud ModelNet40 for different occlusion ratios of the data to measure occlusion robustness of different 3D methods. MVTN achieves 13% better accuracy than PointNet (a robust network) when half of the object is occluded. 60

3.6 **Ablation Study.** We analyze the effect of ablating different MVTN components on test accuracy in ModelNet40. Namely, we observe that using deeper backbone CNNs or a more complex point encoder do not increase the test accuracy. 62

3.7 **Integrating MVTN with Multi-View Networks.** We show overall classification accuracies on ModelNet40 with 12 views on different multi-view networks when fixed views are used versus when MVTN is used. 62

3.8 **Time and Memory Requirements.** We assess the contribution of the MVTN module to the time and memory requirements in the multi-view pipeline. We note that the MVTN’s time and memory requirements are negligible. 63

4.1 **Benchmarking popular DNNs in Semantic Robustness.** We develop the Semantic Robustness Volume Ratio (SRVR) metric to quantify and compare the semantic robustness of well-known DNNs. We see that semantic robustness does not necessarily depend on the accuracy of the DNN. 77

4.2 **Semantic Analysis Techniques:** We compare different approaches to analyse the semantic robustness of DNNs. 78

5.1 **Attack Success Rate (ASR) Comparison:** ASR of adversarial samples generated on three safety-critical applications: YOLOV3 object detection, self-driving, and UAV racing. For detection, we report the average ASR performance across all 12 classes and highlight 3 specific ones. For autonomous driving, we compute the ASR for the three common tasks in CARLA. For UAV racing, we compute ASR for race tracks of varying complexity. Best results are in **bold.** 92
A.1 **Attacking Point Cloud Defenses (DGCNN):** We evaluate untargeted attacks using norm-budgets of $\epsilon_{\infty} = 0.18$ and $\epsilon_{\infty} = 0.45$ with DGCNN [3] as the victim network under different defenses for 3D point clouds. Similar to before, we report attack success rates (higher indicates better attack). Note that both the attacks and evaluations are performed on DGCNN, which has an accuracy of 93.7% without input perturbations (for reference).

A.2 **Attacking Point Cloud Defenses (PointNet++ SSG):** We evaluate untargeted attacks using norm-budgets of $\epsilon_{\infty} = 0.18$ and $\epsilon_{\infty} = 0.45$ with PointNet++ SSG [2] as the victim network under different defenses for 3D point clouds. Similar to before, we report attack success rates (higher indicates better attack). Note that both the attacks and evaluations are performed on PointNet++ SSG, which has an accuracy of 91.5% without input perturbations (for reference).

A.3 **Attacking Point Cloud Defenses (PointNet++ MSG):** We evaluate untargeted attacks using norm-budgets of $\epsilon_{\infty} = 0.18$ and $\epsilon_{\infty} = 0.45$ with PointNet++ MSG [2] as the victim network under different defenses for 3D point clouds. Similar to before, we report attack success rates (higher indicates better attack). Note that both the attacks and evaluations are performed on PointNet++ MSG, which has an accuracy of 91.5% without input perturbations (for reference).

A.4 **Attacking Point Cloud Defenses (PointNet):** We evaluate untargeted attacks using norm-budgets of $\epsilon_{\infty} = 0.18$ and $\epsilon_{\infty} = 0.45$ with PointNet [1] as the victim network under different defenses for 3D point clouds. Similar to before, we report attack success rates (higher indicates better attack). Note that both the attacks and evaluations are performed on PointNet, which has an accuracy of 92.8% without input perturbations (for reference).

A.5 **Soft vs Hard on DGCNN:** study the effect of every bit of the loss on the norms, Attack Success Rate (ASR) and Transferability (TR) under unconstrained setup vs constrained setup in DGCNN [3]. ($\epsilon_{\infty} = 0.18$, $\epsilon_{2} = 2.8$), $\lambda = 1$, $\gamma = 0.5$. Bold numbers are the best.

B.1 **3D Shape Classification on ModelNet40.** We compare MVTN against other methods in 3D classification on ModelNet40 [19]. * indicates results from our rendering setup (differentiable pipeline), while other multi-view results are reported from pre-rendered views. **Bold** denotes the best result in its setup. In brackets, we report the average and standard deviation of four runs.
B.2 **3D Point Cloud Classification on ScanObjectNN.** We compare the performance of MVTN in 3D point cloud classification on three different variants of ScanObjectNN [17]. The variants include object with background, object only, and the hardest variant. * indicates results from our rendering setup (differentiable pipeline), and we report the average and standard deviation of four runs in brackets.

B.3 **3D Shape Retrieval.** We benchmark the shape retrieval capability of MVTN on ModelNet40 [19] and ShapeNet Core55 [16, 20]. MVTN achieves the best retrieval performance among recent state-of-the-art methods on both datasets with only 12 views. In brackets, we report the average and standard deviation of four runs.

B.4 **Effect of Color Selection.** We ablate selecting the color of the object in training our MVTN and when views are fixed in the spherical configuration. Fixed white color is compared to random colors in training. Note how randomizing the color helps in improving the test accuracy on ModelNet40 a little bit.

B.5 **Ablation Study.** We study the effect of ablating different components of MVTN on the test accuracy on ModelNet40. Namely, we observe that using more complex backbone CNNs (like ResNet50 [11]) or a more complex features extractor (like DGCNN [3]) does not increase the performance significantly compared to ResNet18 and PointNet [1] respectively. Furthermore, combining the shape features extractor with the MVCNN [6] in late fusion does not work as well as MVTN with the same architectures. All the reported results are using MVCNN [6] as multi-view network.

D.1 **Special Cases of Generic Adversarial Attacks:** we summarize all the variable substitutions to get common adversarial attacks.

D.2 **Attack Success Rate on Object Detection.** We sample 250 parameters after the training phase of each model and sample a shape from the intended class. We then render an image according to these parameters and run the YOLOV3 detector to obtain a confidence score of the intended class. If this score \( Q \leq \epsilon = 0.3 \), then we consider the attack successful. The success rate is then recorded for that model, while \( u_{std} \) (the mean of standard deviations of each parameter dimension) is recorded for each model.
D.3 ASR for Autonomous Driving (CARLA) and UAV Racing Track Generation (Sim4CV). Each method produces 50 samples and we show the success rate and the mean of the standard deviation per parameter. We set the fooling threshold to 0.6 and 0.7 for autonomous driving and racing track generation respectively.
Chapter 1

Introduction

1.1 Motivation

As a result of recent advances in machine learning and computer vision, deep neural networks (DNNs) are now interleaved with many aspects of our daily lives. DNNs suggest news articles to read and movies to watch, automatically edit our photos and videos, and translate between hundreds of languages. They are also bound to disrupt transportation with autonomous driving slowly becoming a reality. While there are already impressive demos and some successful deployments, safety concerns for boundary conditions persist. While current models work very well on average, they struggle with robustness in certain cases.

Despite its performance, several works show that deep learning algorithms can be susceptible to adversarial attacks. These attacks craft small perturbations to the inputs that push the network to produce incorrect outputs. There is significant progress made in 2D image adversarial attacks, where extensive work shows diverse ways to attack 2D neural networks [21, 22, 23, 24, 25]. In contrast, there is little focus on the 3D robustness of DNNs that operate directly on the 3D domain (i.e. 3D attacks) [4, 26, 27, 5], or 2D DNNs that suffer from 3D physical attacks (i.e. 2.5D attacks) [28, 29]. Furthermore, many of the current 3D understanding methods suffer performance degradation when confronted with realistic transformations such as occlusion and rotation [30]. Our aim in this work is to investigate, analyze, and propose solutions to some of those issues.
1.2 Robustness of 3D Models

1.2.1 3D point Clouds

3D point clouds captured by 3D sensors like LiDAR are now widely processed using deep networks for safety-critical applications, including but not limited to self-driving [31, 32]. However, as we show in this dissertation, 3D deep networks tend to be vulnerable to input perturbations, a fact that increases the risk of using them in such applications. We present a novel approach to attack deep learning algorithms applied to 3D point clouds with a primary focus on attack transferability between networks. Transferability allows an adversary to fool any network, without access to the network’s architecture and is studied extensively in image domain[33, 34, 35]. Clearly, transferable attacks pose a serious security concern, especially in the context of deep learning model deployment. In this dissertation, the goal is to generate adversarial attacks with network transferability, i.e. the attack to a given point cloud is generated using a single and accessible victim network, and the perturbed sample is directly applied to an unseen and inaccessible transfer network. Accessibility here refers to whether the parameters and architecture of the network are known while optimizing the attack (white-box).

1.2.2 Multi-view 2D images

Deep learning techniques have been expanded into 3D vision by operating directly on 3D data represented as point clouds, meshes, or voxels. However, alternative methods represent 3D information through the rendering of multiple 2D views of objects or scenes. Recent advancements in multi-view methods have exhibited exceptional performance in 3D shape classification and segmentation. These approaches leverage 2D convolutional architectures and larger image datasets for pre-training. However, the method for selecting rendering view-points for these techniques remains largely
unexplored. To address this deficiency, we propose using a Multi-View Transformation Network (MVTN) to learn optimal view-points, resulting in the most suitable views for the task and leading to state-of-the-art results in 3D classification and shape retrieval on standard benchmarks. MVTN also improves the robustness of multi-view approaches to rotation and occlusion.

1.3 Physical Robustness of 2D Models

1.3.1 Analysing 2D DNN Classifiers in 3D Setups

In the 2D image domain, adversarial network attacks explore DNN sensitivity and perform gradient updates to derive targeted perturbations [21, 22, 25, 23]. In practice, such pixel attacks are less likely to naturally occur than semantic attacks, such as changes in camera viewpoint and lighting conditions. The literature on semantic attacks (2.5D attacks) is sparser since they are more subtle and challenging to analyze [36, 29, 28]. This is due to the fact that we are unable to distinguish between failure cases that result from the network structure, and learning, or from the data bias [37]. We present a novel approach to finding robust/adversarial regions in the n-dimensional semantic space of 2D CNN classifiers. We use this method to quantify the semantic robustness of popular DNNs on a collected dataset.

1.3.2 Robustness of Vision-based Autonomous Applications

Furthermore, we study 2.5D attacks on advanced DNNs architectures that are used in applications beyond image classification: object detection, autonomous driving, and UAV racing. To this end, we consider environments that are adequately photo-realistic and parameterized by a compact set of variables that have direct semantic meaning (e.g. camera viewpoint, lighting/weather conditions, road layout, etc.). Since the generation process of these environments from their parameters is quite complicated and in general non-differentiable, we treat it as a black-box function that can be
queried but not back-propagated through. We seek to learn an adversary that can produce fooling parameters to construct an environment where the agent (which is also a black-box) fails in its task. Unlike most adversarial attacks that generate sparse instances of failure, our proposed adversary provides a more comprehensive view on how an agent can fail; we learn the distribution of fooling parameters for a particular agent and task and then sample from it. Since Generative Adversarial Networks (GANs [38, 39]) have emerged as a promising family of unsupervised learning techniques that can model high-dimensional distributions, we model our adversary as a GAN, denoted as black-box GAN (BBGAN).

1.4 Presentation and Structure

We start next with Chapter 2 which tackles adversarial attacks on 3D point clouds. In particular, we propose a new pipeline and loss function to perform transferable adversarial perturbations on 3D point clouds. By introducing a data adversarial loss targeting the victim network after reconstructing the perturbed input with a point cloud AE, our approach can be successful in both attacking the victim network and transferring to unseen networks. Since the AE is trained to leverage the point cloud data distribution, incorporating it into the attack strategy enables better transferability to unseen networks. To the best of our knowledge, we are the first to introduce network-transferable adversarial perturbations for 3D point clouds. We perform extensive experiments under constrained norm-budgets to validate the transferability of our attacks. We transfer our attacks between four point cloud networks and show superiority against the state-of-the-art. Furthermore, we demonstrate how our attacks outperform others when targeted by currently available point cloud defenses.

Chapter 3 delves into the paradigm of multi-view for 3D understanding. It addresses a key question about choosing viewpoints for 3D understanding tasks and
proposes a learning framework to learn viewpoints optimized for the task in a fully differentiable pipeline that is trained end-to-end. The proposed pipeline (MVTN) achieves state-of-the-art results in 3D classification and retrieval as well as improves the robustness of 3D shape recognition to rotation and occlusion.

Chapter 4 examines the problem of 3D semantic robustness of 2D CNNs more closely. We analyze popular deep networks from a semantic lens showing unexpected behavior in the 1D and 2D semantic space. Then, we develop a novel bottom-up approach to detect robust/adversarial regions in the semantic space of a DNN, which scales well with increasing dimensionality. The method specifically optimizes for the region’s bounds in semantic space (around a point of interest), such that the continuous region offers semantic parameters that confuse the network. Lastly, we develop a new metric to quantify the semantic robustness of DNNs that we dub Semantic Robustness Volume Ratio (SRVR), and we use it to benchmark popular DNNs on a collected dataset.

In Chapter 5, we formalize adversarial attacks in a more general setup to include both semantic and conventional pixel attacks. We propose BBGAN in order to learn the underlying distribution of semantic adversarial attacks and show promising results on three different safety-critical applications used in autonomous navigation.

Finally, we conclude the dissertation with Appendices A, B, C, and D that include derivations and additional results related to chapters 2, 3, 4, and 5 respectively. Appendix E highlights the published papers resulting from this work and the rest of the Ph.D.
Chapter 2

AdvPC: Transferable Adversarial Perturbations on 3D Point Clouds

Deep neural networks are vulnerable to adversarial attacks, in which imperceptible perturbations to their input lead to erroneous network predictions. This phenomenon has been extensively studied in the image domain and has only recently been extended to 3D point clouds [4, 5]. In this chapter, we present novel data-driven adversarial attacks against 3D point cloud networks. We aim to address the following problems in current 3D point cloud adversarial attacks: they do not transfer well between different networks [4], and they are easy to defend against via simple statistical methods [26]. Figure 2.1 illustrates the concept of transferability. The perturbation generated by our method for a 3D point cloud not only flips the class label of a victim network to a wrong class (i.e. it is adversarial), but it also induces a misclassification for the transfer networks that are not involved in generating the perturbation (i.e. it is transferable).

To this extent, we develop a new point cloud attack (dubbed AdvPC) that exploits the input data distribution. To generate more transferable attacks, we use a point cloud Auto-Encoder (AE), which can effectively reconstruct the unperturbed input after it is perturbed, and then add a data adversarial loss. We optimize the perturbation added to the input to fool the classifier before it passes through the AE (regular adversarial loss) and after it passes through the AE (data adversarial loss). In doing so, the attack tends to be less dependent on the victim network, and generalizes better to different networks. Our attack is dubbed “AdvPC”, and our
Figure 2.1: **Transferable Adversarial Perturbations on 3D point clouds**: Generating adversarial attacks to fool PointNet [1](PN) by perturbing a Table point cloud. The perturbed 3D object not only forces PointNet to predict an incorrect class, but also induces misclassification on other unseen 3D networks (PointNet++ [2], DGCNN [3]) that are not involved in generating the perturbation. Fooling unseen networks poses a threat to 3D deep vision models.

Our attacks surpass state-of-the-art attacks [4, 5] by a large margin (up to 40%) on point cloud networks operating on the standard ModelNet40 dataset [19] and for the same maximum allowed perturbation norms (norm-budgets). AdvPC also increases the ability to break defenses by up to 38% as compared to other baselines on the ModelNet40 dataset. The code is available at `https://github.com/ajhamdi/AdvPC`.

### 2.1 Related Work

#### 2.1.1 Deep Learning for 3D Point Clouds

PointNet [1] paved the way as the first deep learning algorithm to operate directly on 3D point clouds. PointNet computes point features independently, and aggregates them using an order invariant function like max-pooling. An update to this work was PointNet++ [2], where points are aggregated at different 3D scales. Subsequent works...
focused on how to aggregate more local context [41] or on more complex aggregation strategies like RNNs [42, 43]. More recent methods run convolutions across neighbors of points, instead of using point-wise operations [3, 44, 45, 46, 47, 44, 48, 49]. Contrary to PointNet and its variants, these works achieve superior recognition results by focusing on local feature representation. In this paper and to evaluate/validate our adversarial attacks, we use three point-wise networks, PointNet [1] and PointNet++ [2] in single-scale (SSG) and multi-scale (MSG) form, and a Dynamic Graph convolutional Network, DGCNN [3]. We study the sensitivity of each network to adversarial perturbations and show the transferability of AdvPC attacks between the networks.

2.1.2 Adversarial Attacks

Pixel-based Adversarial Attacks. The initial image-based adversarial attack was introduced by Szegedy et al. [21], who cast the attack problem as optimization with pixel perturbations being minimized so as to fool a trained classifier into predicting a wrong class label. Since then, the topic of adversarial attacks has attracted much attention [22, 23, 24, 25, 50]. More recent works take a learning-based approach to the attack [34, 35, 51]. They train a neural network (adversary) to perform the attack and then use the trained adversary model to attack unseen samples. These learning approaches [34, 35, 51] tend to have better transferability properties than the optimizations approaches [22, 23, 24, 25, 50], while the latter tend to achieve higher success rates on the victim networks. As such, our proposed AdvPC attack is a hybrid approach, in which we leverage an AE to capture properties of the data distribution but still define the attack as an optimization for each sample. In doing so, AdvPC captures the merits of both learning and optimization methods to achieve high success rates on the victim networks as well as better transferability to unseen networks.

Adversarial Attacks in 3D. Several adversarial attacks have moved beyond pixel
perturbations to the 3D domain. One line of work focuses on attacking image-based CNNs by changing the 3D parameters of the object in the image, instead of changing the pixels of the image [29, 28, 52]. Recently, Xiang et al. [4] developed adversarial perturbations on 3D point clouds, which were successful in attacking PointNet [1]; however, this approach has two main shortcomings. First, it can be easily defended against by simple statistical operations [26]. Second, the attacks are non-transferable and only work on the attacked network [4, 26]. In contrast, Zheng et al. [27] proposed dropping points from the point cloud using a saliency map, to fool trained 3D deep networks. As compared to [27], our attacks are modeled as an optimization on the additive perturbation variable with a focus on point perturbations instead of point removal. As compared to [4], our AdvPC attacks are significantly more successful against available defenses and more transferable beyond the victim network, since AdvPC leverages the point cloud data distribution through the AE. Concurrent to our work is the work of Tsai et al. [5], in which the attack is crafted with KNN loss to make smooth and natural shapes. The motivation of their work is to craft natural attacks on 3D point clouds that can be 3D-printed into real objects. In comparison, our novel AdvPC attack utilizes the data distribution of point clouds by utilizing an AE to generalize the attack.

Defending Against 3D Point Cloud Attacks. Zhou et al. [26] proposed a Statistical Outlier Removal (SOR) method as a defense against point cloud attacks. SOR uses KNN to identify and remove point outliers. They also propose DUP-Net, which is a combination of their SOR and a point cloud up-sampling network PU-Net [53]. Zhou et al. also proposed removing unnatural points by Simple Random Sampling (SRS), where each point has the same probability of being randomly removed. Adversarial training on the attacked point cloud is also proposed as a mode of defense by [4]. Our attacks surpass state-of-the-art attacks [4, 5] on point cloud networks by a large margin (up to 38%) on the standard ModelNet40 dataset [19]
Figure 2.2: AdvPC Attack Pipeline: We optimize for the constrained perturbation variable $\Delta$ to generate the perturbed sample $X' = X + \Delta$. The perturbed sample fools a trained classifier $F$ (i.e. $F(X')$ is incorrect), and at the same time, if the perturbed sample is reconstructed by an Auto-Encoder (AE) $G$, it too fools the classifier (i.e. $F(G(X'))$ is incorrect). The AdvPC loss for network $F$ is defined in Eq (2.6) and has two parts: network adversarial loss (purple) and data adversarial loss (green).

against the aforementioned defenses [26].

2.2 Methodology

The pipeline of AdvPC is illustrated in Figure 2.2. It consists of an Auto-Encoder (AE) $G$, which is trained to reconstruct 3D point clouds and a point cloud classifier $F$. We seek to find a perturbation variable $\Delta$ added to the input $X$ to fool $F$ before and after it passes through the AE for reconstruction. The setup makes the attack less dependent on the victim network and more dependent on the data. As such, we expect this strategy to generalize to different networks. Next, we describe the main components of our pipeline: 3D point cloud input, AE, and point cloud classifier. Then, we present our attack setup and loss.

2.2.1 AdvPC Attack Pipeline

3D Point Clouds ($X$). We define a point cloud $X \in \mathbb{R}^{N \times 3}$, as a set of $N$ 3D points, where each point $x_i \in \mathbb{R}^3$ is represented by its 3D coordinates $(x_i, y_i, z_i)$.

Point Cloud Networks ($F$). We focus on 3D point cloud classifiers with a feature
max pooling layer as detailed in Eq (2.1), where $h_{\text{mlp}}$ and $h_{\text{conv}}$ are MLP and Convolutional (1 × 1 or edge) layers, respectively. This produces a K-class classifier $F$.

$$F(\mathcal{X}) = h_{\text{mlp}}(\max_{x_i \in \mathcal{X}} \{h_{\text{conv}}(x_i)\})$$ (2.1)

Here, $F : \mathbb{R}^{N \times 3} \rightarrow \mathbb{R}^K$ produces the logits layer of the classifier with size $K$. For our attacks, we take $F$ to be one of the following widely used networks in the literature: PointNet [1], PointNet++ [2] in single-scale form (SSG) and multi-scale form (MSG), and DGCNN [3].

**Point Cloud Auto-Encoder (G).** An AE learns a representation of the data and acts as an effective defense against adversarial attacks. It ideally projects a perturbed point cloud onto the natural manifold of inputs. Any AE architecture in point clouds can be used, but we select the one in [54] because of its simple structure and effectiveness in recovering from adversarial perturbation. The AE $G$ consists of an encoding part, $g_{\text{encode}} : \mathbb{R}^{N \times 3} \rightarrow \mathbb{R}^q$ (similar to Eq (2.1)), and an MLP decoder, $g_{\text{mlp}} : \mathbb{R}^q \rightarrow \mathbb{R}^{N \times 3}$, to produce a point cloud. It can be described formally as: $G(\cdot) = g_{\text{mlp}}(g_{\text{encode}}(\cdot))$. We train the AE with the Chamfer loss as in [54] on the same data used to train $F$, such that it can reliably encode and decode 3D point clouds.

We freeze the AE weights during the optimization of the adversarial perturbation on the input. Since the AE learns how naturally occurring point clouds look like, the gradients updating the attack, which is also tasked to fool the reconstructed sample after the AE, actually become more dependent on the data and less on the victim network. The enhanced data dependency of our attack results in the success of our attacks on unseen transfer networks besides the success on the victim network. As such, the proposed composition allows the crafted attack to successfully attack the victim classifier, as well as, fool transfer classifiers that operate on a similar input data manifold.
2.2.2 AdvPC Attack Loss

Soft Constraint Loss. In AdvPC attacks, like the ones in Figure 2.3, we focus solely on perturbations of the input. We modify each point $x_i$ by a an additive perturbation variable $\delta_i$. Formally, we define the perturbed point set $\mathcal{X}' = \mathcal{X} + \Delta$, where $\Delta \in \mathbb{R}^{N \times 3}$ is the perturbation parameter we are optimizing for. Consequently, each pair $(x_i, x_i')$ are in correspondence. Adversarial attacks are commonly formulated as in Eq (2.2), where the goal is to find an input perturbation $\Delta$ that successfully fools $F$ into predicting an incorrect label $t'$, while keeping $\mathcal{X}'$ and $\mathcal{X}$ close under distance metric $D: \mathbb{R}^{N \times 3} \times \mathbb{R}^{N \times 3} \rightarrow \mathbb{R}$.

$$
\min_{\Delta} \quad D(\mathcal{X}, \mathcal{X}') \quad \text{s.t.} \quad \left[ \arg\max_i F(x'_i) \right] = t' \tag{2.2}
$$

The formulation in Eq (2.2) can describe targeted attacks (if $t'$ is specified before the attack) or untargeted attacks (if $t'$ is any label other than the true label of $\mathcal{X}$). We adopt the following choice of $t'$ for untargeted attacks: $t' = \left[ \arg\max_{i \neq \text{true}} F(x'_i) \right]$. Unless stated otherwise, we primarily use untargeted attacks in this paper. As pointed out in [25], it is difficult to directly solve Eq (2.2). Instead, previous works like [4, 5] have used the well-known C&W formulation, giving rise to the commonly known soft constraint attack: $\min_{\Delta} f_{t'}(F(\mathcal{X}')) + \lambda D(\mathcal{X}, \mathcal{X}')$ where $f_{t'}(F(\mathcal{X}'))$ is the adversarial loss function defined on the network $F$ to move it to label $t'$ as in Eq (2.3).

$$
f_{t'}(F(\mathcal{X}')) = \max \left( \max_{i \neq t'} (F(\mathcal{X}'))_i - F(\mathcal{X}''), 0 \right), \tag{2.3}
$$

where $\kappa$ is a loss margin. The 3D-Adv attack [4] uses $\ell_2$ for $D(\mathcal{X}, \mathcal{X}')$, while the KNN Attack [5] uses Chamfer Distance.

Hard Constraint Loss. An alternative to Eq (2.2) is to put $D(\mathcal{X}, \mathcal{X}')$ as a hard constraint, where the objective can be minimized using Projected Gradient Descent.
Figure 2.3: **Examples of Targeted AdvPC Attacks**: Adversarial attacks are generated for victim networks PointNet, PointNet ++ (MSG/SSG) and DGCNN using AdvPC. The unperturbed point clouds are in black (*top*) while the perturbed examples are in blue (*bottom*). The network predictions are shown under each point cloud.

Using PGD [23, 50] as follows.

\[
\min_\Delta f_t'(F(X')) \quad s.t. \quad D(X, X') \leq \epsilon \tag{2.4}
\]

Using a hard constraint sets a limit to the amount of added perturbation in the attack. This limit is defined by \(\epsilon\) in Eq (2.4), which we call norm-budget in this work. Having this bound ensures a fair comparison between different attack schemes. We compare these schemes by measuring their attack success rate at different levels of norm-budget. Using PGD, the above optimization in Eq (2.4) with \(\ell_p\) distance \(D_{\ell_p}(X, X')\) can be solved by iteratively projecting the perturbation \(\Delta\) onto the \(\ell_p\) sphere of size \(\epsilon_p\) after each gradient step such that: \(\Delta_{t+1} = \Pi_p(\Delta_t - \eta \nabla \Delta_t f_t'(F(X')), \epsilon_p)\). Here, \(\Pi_p(\Delta, \epsilon_p)\) projects the perturbation \(\Delta\) onto the \(\ell_p\) sphere of size \(\epsilon_p\), and \(\eta\) is a step size. The two most commonly used \(\ell_p\) distance metrics in the literature are \(\ell_2\), which measures the energy of the perturbation, and \(\ell_\infty\), which measures the maximum point perturbation of each \(\delta_i \in \Delta\). In our experiments, we choose to use the \(\ell_\infty\) distance defined as \(D_{\ell_\infty}(X, X') = \max_i ||\delta_i||_\infty\). The projection of \(\Delta\) onto the \(\ell_\infty\) sphere of size \(\epsilon_\infty\) is: \(\Pi_{\ell_\infty}(\Delta, \epsilon_\infty) = SAT_{\epsilon_\infty}(\delta_i), \forall \delta_i \in \Delta\), where \(SAT_{\epsilon_\infty}(\delta_i)\) is the element-wise saturation function that takes every element of vector \(\delta_i\) and limits its range to
Data Adversarial Loss. The objectives in Eq (2.2, 2.4) focus solely on the network $F$. We also want to add more focus on the data in crafting our attacks. We do so by fooling $F$ using both the perturbed input $\mathcal{X}'$ and the AE reconstruction $G(\mathcal{X}')$ (see Figure 2.2). Our new objective becomes:

$$
\min_{\Delta} \mathcal{D}(\mathcal{X}, \mathcal{X}') \quad \text{s.t. } [\arg \max_i F(\mathcal{X}')_i] = t'; \ [\arg \max_i F(G(\mathcal{X}'))_i] = t''
$$

(2.5)

Here, $t''$ is any incorrect label $t'' \neq \arg \max_i F(\mathcal{X})_i$ and $t'$ is just like Eq (2.2). The second constraint ensures that the prediction of the perturbed sample after the AE differs from the true label of the unperturbed sample. Similar to Eq (2.2), this objective is hard to optimize, so we follow similar steps as in Eq (2.4) and optimize the following objective for AdvPC using PGD (with $\ell_{\infty}$ as the distance metric):

$$
\min_{\Delta} (1 - \gamma) f_t'(F(\mathcal{X}')) + \gamma f_t'(F(G(\mathcal{X}'))) \quad \text{s.t. } \mathcal{D}_{\ell_{\infty}}(\mathcal{X}, \mathcal{X}') \leq \epsilon_{\infty}
$$

(2.6)

Here, $f$ is as in Eq (2.3), while $\gamma$ is a hyper-parameter that trades off the attack’s success before and after the AE. When $\gamma = 0$, the formulation in Eq (2.6) becomes Eq (2.4). We use PGD to solve Eq (2.6) just like Eq (2.4). We follow the same procedures as in [4] when solving Eq (2.6) by keeping a record of any $\Delta$ that satisfies the constraints in Eq (2.5) and by trying different initializations for $\Delta$.

2.3 Experiments

2.3.1 Setup

Dataset and Networks. We use ModelNet40 [19] to train the classifier network ($F$) and the AE network ($G$), as well as test our attacks. ModelNet40 contains
12,311 CAD models from 40 different classes. These models are divided into 9,843 for training and 2,468 for testing. Similar to previous work [26, 4, 27], we sample 1,024 points from each object. We train the \textbf{F} victim networks: PointNet[1], PointNet++ in both Single-Scale (SSG) and Multi-scale (MSG) [2] settings, and DGCNN [3]. For a fair comparison, we adopt the subset of ModelNet40 detailed in [4] to perform and evaluate our attacks against their work (we call this the attack set). In the attack set, 250 examples are chosen from 10 ModelNet40 classes. We train the \textit{AE} using the full ModelNet40 training set with the Chamfer Distance loss and then fix the \textit{AE} when the attacks are being generated.

\textbf{Adversarial Attack Methods.} We compare AdvPC against the state-of-the-art baselines 3D-Adv [4] and KNN Attack [5]. For all attacks, we use Adam optimizer [55] with learning rate $\eta = 0.01$, and perform 2 different initializations for the optimization of $\Delta$ (as done in [4]). The number of iterations for the attack optimization for all the networks is 200. We set the loss margin $\kappa = 30$ in Eq (2.3) for both 3D-Adv [4] and AdvPC and $\kappa = 15$ for KNN Attack [5] (as suggested in their paper). For other hyperparameters of [4, 5], we follow what is reported in their papers. We pick $\gamma = 0.25$ in Eq (2.6) for AdvPC because it strikes a balance between the success of the attack and its transferability (refer to Section 2.4 for details). In all of the attacks, we follow the same procedure as [4], where the best attack that satisfies the objective during the optimization is reported. We add the hard $\ell_\infty$ projection $\Pi_\infty(\Delta, \epsilon_\infty)$ described in Section 2.2 to all the methods to ensure fair comparison on the same norm-budget $\epsilon_\infty$. We report the best performance of the baselines obtained under this setup.

\textbf{Transferability.} We follow the same setup as [34, 35] by generating attacks using the constrained $\ell_\infty$ metric and measure their success rate at different norm-budgets $\epsilon_\infty$ taken to be in the range [0, 0.75]. This range is chosen because it enables the attacks to reach 100% success on the victim network, as well as offer an opportunity
Figure 2.4: **Transferability Across Different Norm-Budgets**: Here, the victim network is DGCNN [3] and the attacks are optimized using different $\epsilon_\infty$ norm-budgets. We report the attack success on DGCNN and on the transfer networks (PointNet, PointNet++ MSG, and PointNet++ SSG). We note that our AdvPC transfers better to the other networks across different $\epsilon_\infty$ as compared to the baselines 3D-Adv[4] and KNN Attack [5].

For transferability to other networks. We compare AdvPC against the state-of-the-art baselines [4, 5] under these norm-budgets (*e.g.* see Figure 2.4 for attacking DGCNN).

To measure the success of the attack, we compute the percentage of samples out of all attacked samples that the victim network misclassified. We also measure transferability from each victim network to the transfer networks. For each pair of networks, we optimize the attack on one network (victim) and measure the success rate of this optimized attack when applied as input to the other network (transfer). We report these success rates for all network pairs. No defenses are used in the transferability experiment. All the attacks performed in this section are untargeted attacks (following the convention for transferability experiments [4]).

**Attacking the Defenses.** We also analyze the success of our attacks against point cloud defenses. We compare AdvPC attacks and the baselines [4, 5] against several defenses used in the point cloud literature: SOR, SRS, DUP-Net [26], and Adversarial Training [4]. We also add a newly trained AE (different from the one used in the AdvPC attack) to this list of defenses. For SRS, we use a drop rate of 10%, while in SOR, we use the same parameters proposed in [26]. We train DUP-Net on ModelNet40 with an up-sampling rate of 2. For Adversarial Training, all four networks are trained using a mix of the training data of ModelNet40 and adversarial attacks generated by [4].
Table 2.1: **Transferability of Attacks**: We use norm-budgets (max $\ell_\infty$ norm allowed in the perturbation) of $\epsilon_\infty = 0.18$ and $\epsilon_\infty = 0.45$. All the reported results are the untargeted Attack Success Rate (higher numbers are better attacks). **Bold** numbers indicate the most transferable attacks. Our attack consistently achieves better transferability than the other attacks. For reference, the classification accuracies on unperturbed samples for networks PN, PN++(MSG), PN++(SSG) and DGCNN are 92.8%, 91.5%, 91.5%, and 93.7%, respectively.

2.3.2 Results

We present quantitative results that focus on two main aspects. First, we show the transferable power of AdvPC attacks to different point cloud networks. Second, we highlight the strength of AdvPC under different point cloud defenses.

**Transferability.** Table 2.1 reports transferability results for $\epsilon_\infty = 0.18$ and $\epsilon_\infty = 0.45$ and compares AdvPC with the baselines [4, 5]. The value $\epsilon_\infty = 0.18$ is chosen, since it allows the DGCNN attack to reach maximum success, and the value $\epsilon_\infty = 0.45$ is arbitrarily chosen to be midway in the remaining range of $\epsilon_\infty$. It is clear that AdvPC attacks consistently beat the baselines when transferring between networks (up to 40%). Our method shows substantial gains in the case of DGCNN. We also report transferability results for a range of $\epsilon_\infty$ values in Figure 2.4 when the victim network is DGCNN, and the attacks transferred to all other networks. In Appendix A.2, we show the same plots when the victim network is taken to be PN and PN++. To
represent all these transferability curves compactly, we aggregate their results into a Transferability Matrix. Every entry in this matrix measures the transferability from the victim network (row) to the transfer network (column), and it is computed as the average success rate of the attack evaluated on the transfer network across all $\epsilon_\infty$ values. This value reflects how good the perturbation is at fooling the transfer network overall. As such, we advocate the use of the transferability matrix as a standard mode of evaluation for future work on network-transferable attacks. In Figure 2.5, we show the transferability matrices for our attack and the baselines. AdvPC transfers better overall, since it leads to higher (brighter) off-diagonal values in the matrix. Using the average of off-diagonal elements in this matrix as a single scalar measure of transferability, AdvPC achieves 24.9% average transferability, as compared to 11.5% for 3D-Adv [4] and 8.92% for KNN Attack [5]. We note that DGCNN [3] performs best in terms of transferability and is the hardest network to attack (for AdvPC and the baselines).

**Attacking Defenses.** Since DGCNN performs the best in transferability, we use

---

**Figure 2.5:** Transferability Matrix: Visualizing the overall transferability for 3D-Adv [4] (left), KNN Attack [5] (middle), and our AdvPC (right). Elements in the same row correspond to the same victim network used in the attack, while those in the same column correspond to the network that the attack is transferred to. Each matrix element measures the average success rate over the range of $\epsilon_\infty$ for the transfer network. Brighter off-diagonal matrix elements indicate better transferability. The transferability score under each matrix is the average of the off-diagonal matrix values, which summarizes overall transferability for an attack.
Table 2.2: **Attacking Point Cloud Defenses:** We evaluate untargeted attacks using norm-budgets of $\epsilon_\infty = 0.18$ and $\epsilon_\infty = 0.45$ with DGCNN [3] as the victim network under different defenses for 3D point clouds. Similar to before, we report attack success rates (higher indicates better attack). AdvPC consistently outperforms the other attacks [4, 5] for all defenses. Note that both the attacks and evaluations are performed on DGCNN, which has an accuracy of 93.7% without input perturbations (for reference).

<table>
<thead>
<tr>
<th>Defenses</th>
<th>$\epsilon_\infty = 0.18$</th>
<th>$\epsilon_\infty = 0.45$</th>
</tr>
</thead>
<tbody>
<tr>
<td>No defense</td>
<td>100</td>
<td>99.6</td>
</tr>
<tr>
<td>AE (newly trained)</td>
<td>9.2</td>
<td>10.0</td>
</tr>
<tr>
<td>Adv Training [4]</td>
<td>7.2</td>
<td>7.6</td>
</tr>
<tr>
<td>SOR [26]</td>
<td>18.8</td>
<td>17.2</td>
</tr>
<tr>
<td>DUP Net [26]</td>
<td>28</td>
<td>28.8</td>
</tr>
<tr>
<td>SRS [26]</td>
<td>43.2</td>
<td>29.2</td>
</tr>
</tbody>
</table>

it to evaluate the resilience of our AdvPC attacks under different defenses. We use the five defenses described in Section 2.3.1 and report their results in Table 2.2. Our attack is more resilient than the baselines against all defenses. We note that the AE defense is very strong against all attacks compared to other defenses [26], which explains why AdvPC works very well against other defenses and transfers well to unseen networks. We also observe that our attack is strong against simple statistical defenses like SRS (38% improvement over the baselines). We report results for other victim networks (PN and PN++) in Appendix A.3, where AdvPC shows superior performance against the baselines under these defenses.

2.4 Analysis

We perform several analytical experiments to further explore the results obtained in Section 2.3.2. We first study the effect of different factors that play a role in the transferability of our attacks. We also show some interesting insights related to the sensitivity of point cloud networks and the effect of the AE on the attacks. All the analysis results are reported in Appendix A.4.
Chapter 3

MVTN: Multi-View Transformation Network for 3D Shape Recognition

The utilization of deep learning techniques in the 2-dimensional domain has led to its expansion into the 3-dimensional vision domain. In the 3D realm, deep networks have demonstrated remarkable results in classification, segmentation, and detection tasks by operating directly on 3D data, which is commonly represented as point clouds [1, 2, 3], meshes [56, 57], or voxels [58, 59, 60]. However, alternative methods exist that involve representing 3D information through the rendering of multiple 2D views of objects or scenes [6]. These multi-view techniques align with the human approach, as the human visual system is exposed to streams of rendered images rather than complex 3D representations.

Recent advancements in multi-view methods have exhibited exceptional performance, achieving state-of-the-art results in 3D shape classification and segmentation [8, 7, 61, 62, 63]. These approaches bridge the divide between 2D and 3D learning by solving 3D tasks through the utilization of 2D convolutional architectures. These methods render multiple views for a specific 3D shape and leverage the rendered images to solve the task at hand. As a result, they build upon recent developments in 2D image deep learning and leverage larger image datasets for pre-training (e.g. ImageNet [12]) to counteract the general scarcity of labeled 3D datasets. However, the method for selecting rendering view-points for these techniques remains largely unexplored. Current methods rely on heuristics such as random sampling in scenes [61] or predefined canonical view-points in oriented datasets [7], without evidence to
Figure 3.1: **Multi-View Transformation Network (MVTN)**. We propose a differentiable module that predicts the best view-points for a task-specific multi-view network. MVTN is trained jointly with this network without any extra training supervision while improving the performance of 3D classification and shape retrieval.

support their empirical superiority.

To address this deficiency, we propose using a Multi-View Transformation Network (MVTN) to learn optimal view-points. The MVTN learns to regress view-points, renders those views with a differentiable renderer, and trains the downstream task-specific network in an end-to-end fashion, resulting in the most suitable views for the task (see Figure 3.1). Combining MVTN with multi-view approaches leads to state-of-the-art results in 3D classification and shape retrieval on standard benchmarks ModelNet40 [19], ShapeNet Core55 [16, 20], and ScanObjectNN [17]. Additional analysis shows that MVTN improves the robustness of multi-view approaches to rotation and occlusion, making MVTN more practical for realistic scenarios, where 3D models are not perfectly aligned or partially cropped.
3.1 Related Work

**Multi-View 3D Shape Classification.** The first work on using 2D images to recognize 3D objects was proposed by Bradski et al. [64]. Twenty years later and after the success of deep learning in 2D vision tasks, MVCNN [6] emerged as the first use of deep 2D CNNs for 3D object recognition. The original MVCNN uses max pooling to aggregate features from different views. Several follow-up works propose different strategies to assign weights to views to perform weighted average pooling of view-specific features [65, 66, 67, 68]. RotationNet [8] classifies the views and the object jointly. Equivariant MV-Network [69] uses a rotation equivariant convolution operation on multi-views by utilizing rotation group convolutions [70]. The more recent work of ViewGCN [7] utilizes dynamic graph convolution operations to adaptively pool features from different fixed views for the task of 3D shape classification. All these previous methods rely on fixed rendered datasets of 3D objects. The work of [68] attempts to select views adaptively through reinforcement learning and RNNs, but this comes with limited success and an elaborate training process. In this paper, we propose a novel MVTN framework for predicting optimal view-points in a multi-view setup. This is done by jointly training MVTN with a multi-view task-specific network, without the need for any extra supervision nor adjustment to the learning process.

**3D Shape Retrieval.** Early methods in the literature compare the distribution of hand-crafted descriptors to retrieve similar 3D shapes. Those shape signatures could either represent geometric [71] or visual [72] cues. Traditional geometric methods would estimate distributions of certain characteristics (e.g., distances, angles, areas, or volumes) to measure the similarity between shapes [73, 74, 75]. Gao et al. [76] use multiple camera projections, and Wu et al. [77] use a voxel grid to extract analogous model-based signatures. Su et al. [6] introduce a deep learning pipeline for multi-view classification, with aggregated features achieving high retrieval performance. They use
To learn adaptive scene parameters $u$ that maximize the performance of a multi-view network $C$ for every 3D object shape $S$, we use a differentiable renderer $R$. MVTN extracts coarse features from $S$ by a point encoder and regresses the adaptive scene parameters for that object. In this example, the parameters $u$ are the azimuth and elevation angles of cameras pointing towards the center of the object. The MVTN pipeline is optimized end-to-end for the task loss.

a low-rank Mahalanobis metric atop extracted multi-view features to improve retrieval performance. This seminal work on multi-view learning is extended for retrieval with volumetric-based descriptors [78], hierarchical view-group architectures [67], and triplet-center loss [79]. Jiang et al. [80] investigate better views for retrieval using many loops of circular cameras around the three principal axes. However, these approaches consider fixed camera view-points compared to MVTN’s learnable ones.

3.2 Methodology

We illustrate our proposed multi-view pipeline using MVTN in Figure 3.2. MVTN is a generic module that learns camera view-point transformations for specific 3D multi-view tasks, e.g., 3D shape classification. In this section, we review a generic framework for common multi-view pipelines, introduce MVTN details, and present an integration of MVTN for 3D shape classification and retrieval.

3.2.1 Overview of Multi-View 3D Recognition

3D multi-view recognition defines $M$ different images $\{x_i\}_{i=1}^M$ rendered from multiple view-points of the same shape $S$. The views are fed into the same backbone network
f that extracts discriminative features per view. These features are then aggregated
among views to describe the entire shape and used for downstream tasks such as
classification or retrieval. Specifically, a multi-view network $C$ with parameters $\theta_C$
operates on an input set of images $X \in \mathbb{R}^{M \times h \times w \times c}$ to obtain a softmax probability
vector for the shape $S$.

**Training Multi-View Networks.** The simplest deep multi-view classifier is
MVCNN, where $C = \text{MLP}(\max_i f(x_i))$ with $f : \mathbb{R}^{h \times w \times c} \to \mathbb{R}^d$ being a 2D CNN
backbone (e.g. ResNet [11]) applied individually on each rendered image. A more
recent method like ViewGCN would be described as $C = \text{MLP}(\text{cat}_{\text{GCN}}(f(x_i)))$,
where $\text{cat}_{\text{GCN}}$ is an aggregation of views' features learned from a graph convolutional
network. In general, learning a task-specific multi-view network on a labeled 3D
dataset is formulated as:

$$
\arg \min_{\theta_C} \sum_n L \left( C(X_n) , y_n \right)
= \arg \min_{\theta_C} \sum_n L \left( C(R(S_n, u_0)) , y_n \right),
$$

where $L$ is a task-specific loss defined over $N$ 3D shapes in the dataset, $y_n$ is the label
for the $n^{th}$ 3D shape $S_n$, and $u_0 \in \mathbb{R}^\tau$ is a set of $\tau$ fixed scene parameters for the
entire dataset. These parameters represent properties that affect the rendered image,
including camera view-point, light, object color, and background. $R$ is the renderer
that takes as input a shape $S_n$ and the parameters $u_0$ to produce $M$ multi-view
images $X_n$ per shape. In our experiments, we choose the scene parameters $u$ to be
the azimuth and elevation angles of the camera view-points pointing towards the
object center, thus setting $\tau = 2M$.

**Canonical Views.** Previous multi-view methods rely on scene parameters $u_0$ that
are pre-defined for the entire 3D dataset. In particular, the fixed camera view-points
are usually selected based on the alignment of the 3D models in the dataset. The most
common view configurations are circular that aligns view-points on a circle around the object [6, 65] and spherical that aligns equally spaced view-points on a sphere surrounding the object [7, 8]. Fixing those canonical views for all 3D objects can be misleading for some classes. For example, looking at a bed from the bottom could confuse a 3D classifier. In contrast, MVTN learns to regress per-shape view-points, as illustrated in Figure 3.3.

### 3.2.2 Multi-View Transformation Network (MVTN)

Previous multi-view methods take the multi-view image $X$ as the only representation for the 3D shape, where $X$ is rendered using fixed scene parameters $u_0$. In contrast, we consider a more general case, where $u$ is variable yet within bounds $\pm u_{\text{bound}}$. Here, $u_{\text{bound}}$ is positive and it defines the permissible range for the scene parameters. We set $u_{\text{bound}}$ to $180^\circ$ and $90^\circ$ for each azimuth and elevation angle.
Differentiable Renderer. A renderer $\mathbf{R}$ takes a 3D shape $\mathbf{S}$ (mesh or point cloud) and scene parameters $\mathbf{u}$ as inputs, and outputs the corresponding $M$ rendered images $\{x_i\}_{i=1}^M$. Since $\mathbf{R}$ is differentiable, gradients $\frac{\partial x_i}{\partial \mathbf{u}}$ can propagate backward from each rendered image to the scene parameters, thus establishing a framework that suits end-to-end deep learning pipelines. When $\mathbf{S}$ is represented as a 3D mesh, $\mathbf{R}$ has two components: a rasterizer and a shader. First, the rasterizer transforms meshes from the world to view coordinates given the camera view-point and assigns faces to pixels. Using these face assignments, the shader creates multiple values for each pixel then blends them. On the other hand, if $\mathbf{S}$ is represented by a 3D point cloud, $\mathbf{R}$ would use an alpha-blending mechanism instead [81]. Figure 3.3 and Figure 3.4 illustrate examples of mesh and point cloud renderings used in MVTN.

View-Points Conditioned on 3D Shape. We design $\mathbf{u}$ to be a function of the 3D shape by learning a Multi-View Transformation Network (MVTN), denoted as $\mathbf{G} \in \mathbb{R}^{P \times 3} \rightarrow \mathbb{R}^r$ and parameterized by $\theta_G$, where $P$ is the number of points sampled from shape $\mathbf{S}$. Unlike Eq (3.1) that relies on constant rendering parameters, MVTN predicts $\mathbf{u}$ adaptively for each object shape $\mathbf{S}$ and is optimized along with the classifier $\mathbf{C}$. The pipeline is trained end-to-end to minimize the following loss on a dataset of $N$ objects:

$$\arg\min_{\theta_C, \theta_G} \sum_{n}^N L \left( \mathbf{C} \left( \mathbf{R}(\mathbf{S}_n, \mathbf{u}_n) \right), y_n \right),$$

(3.2)

s. t. $\mathbf{u}_n = \mathbf{u}_{\text{bound}} \cdot \tanh \left( \mathbf{G}(\mathbf{S}_n) \right)$

Here, $\mathbf{G}$ encodes a 3D shape to predict its optimal view-points for the task-specific multi-view network $\mathbf{C}$. Since the goal of $\mathbf{G}$ is only to predict view-points and not classify objects (as opposed to $\mathbf{C}$), its architecture is designed to be simple and light-weight. As such, we use a simple point encoder (e.g. shared MLP as in PointNet [1]) that processes $P$ points from $\mathbf{S}$ and produces coarse shape features of dimension $b$. Then, a shallow MLP regresses the scene parameters $\mathbf{u}_n$ from the global shape features. To force the predicted parameters $\mathbf{u}$ to be within a permissible range
Figure 3.4: **Multi-View Point Cloud Renderings.** We show some examples of point cloud renderings used in our pipeline. Note how point cloud renderings offer more information about content hidden from the camera view-point (*e.g.* car wheels from the occluded side), which can be useful for recognition.

$\pm \mathbf{u}_{\text{bound}}$, we use a hyperbolic tangent function scaled by $\mathbf{u}_{\text{bound}}$.

**MVTN for 3D Shape Classification.** To train MVTN for 3D shape classification, we define a cross-entropy loss in Eq (3.2), yet other losses and regularizers can be used here as well. The multi-view network ($\mathbf{C}$) and the MVTN ($\mathbf{G}$) are trained jointly on the same loss. One merit of our multi-view pipeline is its ability to seamlessly handle 3D point clouds, which is absent in previous multi-view methods. When $\mathbf{S}$ is a 3D point cloud, we simply define $\mathbf{R}$ as a differentiable point cloud renderer.

**MVTN for 3D Shape Retrieval.** The shape retrieval task is defined as follows: given a query shape $\mathbf{S}_q$, find the most similar shapes in a broader set of size $N$. For this task, we follow the retrieval setup of MVCNN [6]. In particular, we consider the deep feature representation of the last layer before the classifier in $\mathbf{C}$. We project those features into a more expressive space using LFDA reduction [82] and consider the reduced feature as the signature to describe a shape. At test time, shape signatures are used to retrieve (in order) the most similar shapes in the training set.
3.3 Experiments

We evaluate MVTN for the tasks of 3D shape classification and retrieval on ModelNet40 [19], ShapeNet Core55 [16], and the more realistic ScanObjectNN [17].

3.3.1 Datasets

ModelNet40. ModelNet40 [19] is composed of 12,311 3D objects (9,843/2,468 in training/testing) labelled with 40 object classes. Since we render 3D models in the forward pass, we limit the number of triangles in the meshes due to hardware constraints. In particular, we simplify the meshes to 20k vertices using the official Blender API [13, 83].

ShapeNet Core55. ShapeNet Core55 is a subset of ShapeNet [16] comprising 51,162 3D mesh objects labelled with 55 object classes. The training, validation, and test sets consist of 35764, 5133, and 10265 shapes, respectively. It is designed for the shape retrieval challenge SHREK [20].

ScanObjectNN. ScanObjectNN [17] is a recently released point cloud dataset for 3D classification that is more realistic and challenging than ModelNet40, since it includes background and considers occlusions. The dataset is composed of 2902 point clouds divided into 15 object categories. We consider its three main variants: object only, object with background, and the hardest perturbed variant (PB_T50_RS variant). These variants are used in the 3D Scene Understanding Benchmark associated with the ScanObjectNN dataset. This dataset offers a more challenging setup than ModelNet40 and tests the generalization capability of 3D deep learning model in more realistic scenarios.

3.3.2 Metrics

Classification Accuracy. The standard evaluation metric in 3D classification is accuracy. We report overall accuracy (percentage of correctly classified test samples)
and average per-class accuracy (mean of all true class accuracies).

**Retrieval mAP.** Shape retrieval is evaluated by mean Average Precision (mAP) over test queries. For every query shape \( S_q \) from the test set, AP is defined as 
\[
AP = \frac{1}{GTP} \sum_{n}^{N} \frac{1(S_n)}{n},
\]
where \( GTP \) is the number of ground truth positives, \( N \) is the size of the ordered training set, and \( 1(S_n) = 1 \) if the shape \( S_n \) is from the same class label of query \( S_q \). We average the retrieval AP over the test set to measure retrieval mAP.

### 3.3.3 Baselines

**Voxel Networks.** We choose VoxNet [58], DLAN [84], and 3DShapeNets [19] as baselines that use voxels.

**Point Cloud Networks.** We select PointNet [1], PointNet++ [2], DGCNN [3], PVNet [85], and KPConv [86] as baselines that use point clouds. These methods leverage different convolution operators on point clouds by aggregating local and global point information.

**Multi-view Networks.** We compare against MVCNN [6], RotationNet [8], GVCNN [67] and ViewGCN [7] as representative multi-view methods. These methods are limited to meshes, pre-rendered from canonical view-points.

### 3.3.4 MVTN Details

**Rendering.** We choose the differentiable mesh and point cloud renderers \( R \) from Pytorch3D [87] in our pipeline for their speed and compatibility with Pytorch libraries [88]. We show examples of the rendered images for meshes (Figure 3.3) and point clouds (Figure 3.4). Each rendered image has a size of 224×224. For ModelNet40, we use the differentiable mesh renderer. We direct the light randomly and assign a random color for the object for augmentation purposes in training. In testing, we keep a fixed light pointing towards the object center and color the object white for stable
<table>
<thead>
<tr>
<th>Method</th>
<th>Data Type</th>
<th>Classification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(Per-Class) (Overall)</td>
</tr>
<tr>
<td>VoxNet [58]</td>
<td>Voxels</td>
<td>83.0  85.9</td>
</tr>
<tr>
<td>PointNet [1]</td>
<td>Points</td>
<td>86.2  89.2</td>
</tr>
<tr>
<td>PointNet++ [2]</td>
<td>Points</td>
<td>-     91.9</td>
</tr>
<tr>
<td>PointCNN [44]</td>
<td>Points</td>
<td>88.1  91.8</td>
</tr>
<tr>
<td>DGCNN [3]</td>
<td>Points</td>
<td>90.2  92.2</td>
</tr>
<tr>
<td>SGAS [89]</td>
<td>Points</td>
<td>-     93.2</td>
</tr>
<tr>
<td>KPConv[86]</td>
<td>Points</td>
<td>-     92.9</td>
</tr>
<tr>
<td>PTransformer[90]</td>
<td>Points</td>
<td><strong>90.6</strong>  93.7</td>
</tr>
<tr>
<td>MVCNN [6]</td>
<td>12 Views</td>
<td>90.1  90.1</td>
</tr>
<tr>
<td>GVCNN [67]</td>
<td>12 Views</td>
<td>90.7  93.1</td>
</tr>
<tr>
<td>ViewGCN [7]</td>
<td>20 Views</td>
<td><strong>96.5</strong>  97.6</td>
</tr>
<tr>
<td>ViewGCN [7]*</td>
<td>12 views</td>
<td>90.7  93.0</td>
</tr>
<tr>
<td>ViewGCN [7]*</td>
<td>20 views</td>
<td>91.3  93.3</td>
</tr>
<tr>
<td>MVTN (ours)*</td>
<td>12 Views</td>
<td>92.0  <strong>93.8</strong></td>
</tr>
<tr>
<td>MVTN (ours)*</td>
<td>20 Views</td>
<td><strong>92.2</strong>  93.5</td>
</tr>
</tbody>
</table>

Table 3.1: **3D Shape Classification on ModelNet40.** We compare MVTN against other methods in 3D classification on ModelNet40 [19]. * indicates results from our rendering setup (differentiable pipeline), while other multi-view results are reported from pre-rendered views. **Bold** denotes the best result in its setup.

Performance. For ShapeNet Core55 and ScanObjectNN, we use the differentiable point cloud renderer using 2048 and 5000 points, respectively. Point cloud rendering offers a light alternative to mesh rendering when the mesh contains a large number of faces that hinders training the MVTN pipeline.

**View-Point Prediction.** As shown in Eq (3.2), the MVTN $G$ network learns to predict the view-points directly (*MVTN-direct*). Alternatively, MVTN can learn relative offsets w.r.t. initial parameters $u_0$. In this case, we concatenate the point features extracted in $G$ with $u_0$ to predict the offsets to apply on $u_0$. The learned view-points $u_n$ in Eq (3.2) are defined as: $u_n = u_0 + u_{\text{bound}} \cdot \tanh(G(u_0, S_n))$. We take $u_0$ to be the circular or spherical configurations commonly used in multi-view
<table>
<thead>
<tr>
<th>Method</th>
<th>OBJ_BG</th>
<th>OBJ_ONLY</th>
<th>Hardest</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DFMV [91]</td>
<td>68.2</td>
<td>73.8</td>
<td>63.0</td>
</tr>
<tr>
<td>PointNet [1]</td>
<td>73.3</td>
<td>79.2</td>
<td>68.0</td>
</tr>
<tr>
<td>SpiderCNN [92]</td>
<td>77.1</td>
<td>79.5</td>
<td>73.7</td>
</tr>
<tr>
<td>PointNet ++ [2]</td>
<td>82.3</td>
<td>84.3</td>
<td>77.9</td>
</tr>
<tr>
<td>PointCNN [44]</td>
<td>86.1</td>
<td>85.5</td>
<td>78.5</td>
</tr>
<tr>
<td>DGCNN [3]</td>
<td>82.8</td>
<td>86.2</td>
<td>78.1</td>
</tr>
<tr>
<td>SimpleView [93]</td>
<td>-</td>
<td>-</td>
<td>79.5</td>
</tr>
<tr>
<td>BGA-DGCNN [17]</td>
<td>-</td>
<td>-</td>
<td>79.7</td>
</tr>
<tr>
<td>BGA-PN++ [17]</td>
<td>-</td>
<td>-</td>
<td>80.2</td>
</tr>
<tr>
<td>MVTN (ours)</td>
<td>92.6</td>
<td>92.3</td>
<td>82.8</td>
</tr>
</tbody>
</table>

Table 3.2: 3D Point Cloud Classification on ScanObjectNN. We compare the performance of MVTN in 3D point cloud classification on three different variants of ScanObjectNN [17]. The variants include object with background, object only, and the hardest variant.

classification pipelines [6, 8, 7]. We refer to these learnable variants as MVTN-circular and MVTN-spherical, accordingly. For MVTN-circular, the initial elevations for the views are 30°, and the azimuth angles are equally distributed over 360° following [6]. For MVTN-spherical, we follow the method from [94] that places equally-spaced view-points on a sphere for an arbitrary number of views, which is similar to the “dodecahedral” configuration in ViewGCN.

**Architecture.** We select MVCNN [6], RotationNet [8], and the more recent ViewGCN [7] as our multi-view networks of choice in the MVTN pipeline. In our experiments, we select PointNet [1] as the 3D point encoder network G and experiment with DGCNN in Section 3.5.1. We sample $P = 2048$ points from each mesh as input to the point encoder and use a 5-layer MLP for the regression network, which takes as input the point features extracted by the point encoder of size $b = 40$. All MVTN variants and the baseline multi-view networks use ResNet-18 [11] pre-trained on ImageNet [12] for the multi-view backbone in $C$, with output features of size $d = 1024$. The main classification and retrieval results are based on
Table 3.3: **3D Shape Retrieval.** We benchmark the shape retrieval mAP of MVTN on ModelNet40 [19] and ShapeNet Core55 [16, 20]. MVTN achieves the best retrieval performance among recent state-of-the-art methods on both datasets with only 12 views.

MVFTN-spherical with ViewGCN [7] as the multi-view network C, unless otherwise specified as in Section 3.4.3 and Section 3.5.1.

**Training Setup.** To avoid gradient instability introduced by the renderer, we use gradient clipping in the MVTN network G. We clip the gradient updates such that the $\ell_2$ norm of the gradients does not exceed 30. We use a learning rate of 0.001 but refrain from fine-tuning the hyper-parameters introduced in MVCNN [6] and View-GCN [7]. More details about the training procedure are in Appendix B.1.

### 3.4 Results

The main results of MVTN are summarized in Tables 3.1, 3.2, 3.3 and 3.4. We achieve state-of-the-art performance in 3D classification on ScanObjectNN by a large margin (up to 6%) and achieve a competitive test accuracy of 93.8% on ModelNet40.
Figure 3.5: **Qualitative Examples for Object Retrieval**: (left): we show some query objects from the test set. (right): we show top five retrieved objects by our MVTN from the training set. Images of negative retrieved objects are framed.

On shape retrieval, we achieve state-of-the-art performance on both ShapeNet Core55 (82.9 mAP) and ModelNet40 (92.9 mAP). Following the common practice, we report the best results out of four runs in benchmark tables, but detailed results are in Appendix B.2.

### 3.4.1 3D Shape Classification

Table 3.1 compares the performance of MVTN against other methods on ModelNet40 [19]. Our MVTN achieves a competitive test accuracy of 93.8% compared to all previous methods. ViewGCN [7] achieves higher classification performance by relying on higher quality images from a more advanced yet non-differentiable OpenGL [99] renderer. For a fair comparison, we report with an * the performance of ViewGCN using images generated by the renderer used in MVTN. Using the same rendering process, regressing views with MVTN improves the classification performance of the baseline ViewGCN at 12 and 20 views. We believe future advances in differentiable rendering would bridge the gap between our rendered images and the original high-
Table 3.4: **Rotation Robustness on ModelNet40.** At test time, we randomly rotate objects in ModelNet40 around the Y-axis (gravity) with different ranges and report the overall accuracy. MVTN displays strong robustness to such Y-rotations.

<table>
<thead>
<tr>
<th>Method</th>
<th>Rotation Perturbations Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0°</td>
</tr>
<tr>
<td>PointNet [1]</td>
<td>88.7</td>
</tr>
<tr>
<td>PointNet ++ [2]</td>
<td>88.2</td>
</tr>
<tr>
<td>RSCNN [30]</td>
<td>90.3</td>
</tr>
<tr>
<td>MVTN (ours)</td>
<td><strong>91.7</strong></td>
</tr>
</tbody>
</table>

quality pre-rendered ones.

Table 3.2 reports the classification accuracy of a 12 view MVTN on the realistic ScanObjectNN benchmark [17]. MVTN improves performance on different variants of the dataset. The most difficult variant of ScanObjectNN (PB_T50_RS) includes challenging scenarios of objects undergoing translation and rotation. Our MVTN achieves state-of-the-art results (+2.6%) on this variant, highlighting the merits of MVTN for realistic 3D point cloud scans. Also, note how adding background points (in OBJ_BG) does not hurt MVTN, contrary to most other classifiers.

### 3.4.2 3D Shape Retrieval

Table 3.3 reports the retrieval mAP of MVTN compared with recent methods on ModelNet40 [19] and ShapeNet Core55 [16]. The results of the latter methods are taken from [80, 7, 85]. MVTN achieves state-of-the-art retrieval performance (92.9% mAP) on ModelNet40. It also improves the state-of-the-art by a large margin in ShapeNet, while only using 12 views. It is important to note that the baselines in Table 3.3 include strong and recent methods trained specifically for retrieval, such as MLVCNN [80]. Figure 3.5 shows qualitative examples of objects retrieved using MVTN.
3.4.3 Rotation Robustness

A common practice in 3D shape classification literature is to test the robustness of trained models to perturbations at test time. Following the same setup as [30, 100], we perturb the shapes with random rotations around the Y-axis (gravity-axis) contained within $\pm 90^\circ$ and $\pm 180^\circ$. We repeat the inference ten times for each setup and report the average performance in Table 3.4. The MVTN-circular variant (with MVCNN) reaches state-of-the-art performance in rotation robustness (91.2% test accuracy) compared to more advanced methods trained in the same setup. The baseline RSCNN [30] is a strong baseline designed to be invariant to translation and rotation. In contrast, MVTN is learned in a simple setup with MVCNN without targeting rotation invariance.

3.4.4 Occlusion Robustness

To test the usefulness of MVTN in a realistic scenario, we investigate the common problem of occlusion in 3D computer vision, especially in 3D point cloud scans. Various factors lead to occlusion, including the view angle to the object, the sensor’s sampling density (e.g. LiDAR), or the presence of noise in the sensor. In such realistic scenarios, deep learning models typically fail. To quantify this occlusion effect due to the viewing angle of the 3D sensor in our setup of 3D classification, we simulate realistic occlusion by cropping the object from canonical directions. We train PointNet [1], DGCNN [3], and MVTN on the ModelNet40 point cloud dataset. Then, at test time, we crop a portion of the object (from 0% occlusion ratio to 100%) along the $\pm X$, $\pm Y$, and $\pm Z$ directions. Figure 3.6 shows examples of this occlusion effect with different occlusion ratios. In all robustness experiments, the studied transformations (rotation or occlusion) happen only in test time. All the methods compared, including MVTN, are trained naturally without any augmentation by those transformations. We report the average test accuracy of the six cropping directions for the baselines and
Figure 3.6: **Occlusion of 3D Objects**: We simulate realistic occlusion scenarios in 3D point clouds by cropping a percentage of the object along canonical directions. Here, we show an object occluded with different ratios and from different directions.

<table>
<thead>
<tr>
<th>Method</th>
<th>Occlusion Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>PointNet [1]</td>
<td>89.1</td>
</tr>
<tr>
<td>DGCNN [3]</td>
<td>92.1</td>
</tr>
<tr>
<td>MVTN (ours)</td>
<td>92.3</td>
</tr>
</tbody>
</table>

Table 3.5: **Occlusion Robustness of 3D Methods**. We report the test accuracy on point cloud ModelNet40 for different occlusion ratios of the data to measure occlusion robustness of different 3D methods. MVTN achieves 13% better accuracy than PointNet (a robust network) when half of the object is occluded.

MVTN in Table 3.5. Note how MVTN achieves high test accuracy even when large portions of the object are cropped. Interestingly, MVTN outperforms PointNet [1] by 13% in test accuracy when half of the object is occluded. This result is significant, given that PointNet is well-known for its robustness [1, 101].
Figure 3.7: **Effect of the Number of Views.** We plot the test accuracy vs. the number of views (M) used to train MVCNN on fixed, random, and learned MVTN view configurations. We observe a consistent 2% improvement with MVTN over a variety of views.

### 3.5 Analysis and Insights

#### 3.5.1 Ablation Study

This section performs a comprehensive ablation study on the different components of MVTN and their effect on the overall test accuracy on ModelNet40 [19].

**Number of Views.** We study the effect of the number of views $M$ on the performance of MVCNN when using fixed views (circular/spherical), learned views (MVTN), and random views. The experiments are repeated four times, and the average test accuracies with confidence intervals are shown in Figure 3.7. The plots show how learned MVTN-spherical achieves consistently superior performance across a different number of views.

**Choice of Backbone and Point Encoders.** In all of our main MVTN experiments, we use ResNet-18 as our backbone and PointNet as the point feature extractor. However, different choices could be made for both. We explore using DGCNN [3] as an alternative point encoder and ResNet-34 as an alternative 2D backbone in
Table 3.6: Ablation Study. We analyze the effect of ablating different MVTN components on test accuracy in ModelNet40. Namely, we observe that using deeper backbone CNNs or a more complex point encoder do not increase the test accuracy.

<table>
<thead>
<tr>
<th>Backbone Network</th>
<th>Point Encoder</th>
<th>MVTN Setup</th>
<th>Results Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>PointNet</td>
<td>circular</td>
<td>92.83 ± 0.06</td>
</tr>
<tr>
<td></td>
<td>DGCNN</td>
<td>spherical</td>
<td>93.41 ± 0.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>circular</td>
<td>93.03 ± 0.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>spherical</td>
<td>93.26 ± 0.04</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>PointNet</td>
<td>circular</td>
<td>92.72 ± 0.16</td>
</tr>
<tr>
<td></td>
<td>DGCNN</td>
<td>spherical</td>
<td>92.83 ± 0.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>circular</td>
<td>92.72 ± 0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>spherical</td>
<td>92.63 ± 0.15</td>
</tr>
</tbody>
</table>

Table 3.7: Integrating MVTN with Multi-View Networks. We show overall classification accuracies on ModelNet40 with 12 views on different multi-view networks when fixed views are used versus when MVTN is used.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed views</td>
<td>90.4</td>
<td>91.6</td>
<td>93.0</td>
<td></td>
</tr>
<tr>
<td>with MVTN</td>
<td>92.6</td>
<td>93.2</td>
<td>93.8</td>
<td></td>
</tr>
</tbody>
</table>

ViewGCN. We report all MVTN ablation results in Table 3.6. We observe diminishing returns for making the CNN backbone and the shape feature extractor more complex in the MVTN setup, which justifies using the simpler combination in our main experiments.

Choice of Multi-View Network. MVTN integrates smoothly with different multi-view networks and always leads to performance boost. In Table 3.7, we show the overall accuracies (averaged over four runs) on ModelNet40 of 12 views when fixed views are used versus when MVTN is used with different multi-view networks.

Other Factors Affecting MVTN. We study the effect of the light direction in the renderer, the camera’s distance to the object, and the object’s color. We also study
Table 3.8: **Time and Memory Requirements.** We assess the contribution of the MVTN module to the time and memory requirements in the multi-view pipeline. We note that the MVTN’s time and memory requirements are negligible.

the transferability of the learned views from one multi-view network to another, and the performance of MVTN variants. More details are provided in Appendix B.3.

### 3.5.2 Time and Memory Requirements

We compare the time and memory requirements of different parts of our 3D recognition pipeline. We record the number of floating-point operations (GFLOPs) and the time of a forward pass for a single input sample. In Table 3.8, MVTN contributes negligibly to the time and memory requirements of the multi-view networks.
Chapter 4

Towards Analyzing Semantic Robustness of Deep Neural Networks

Despite the impressive performance of DNNs on various vision tasks, they still exhibit erroneous high sensitivity toward semantic primitives (e.g., object pose). In this chapter, we propose a theoretically grounded analysis for DNN robustness in the semantic space. We qualitatively analyze different DNNs’ semantic robustness by visualizing the DNN global behavior as semantic maps and observe interesting behavior of some DNNs. Since generating these semantic maps does not scale well with the dimensionality of the semantic space, we develop a bottom-up approach to detect robust regions of DNNs. To achieve this, we formalize the problem of finding robust semantic regions of the network as optimizing integral bounds and we develop expressions for update directions of the region bounds. We use our developed formulations to quantitatively evaluate the semantic robustness of different popular network architectures. We show through extensive experimentation that several networks, while trained on the same dataset and enjoying comparable accuracy, do not necessarily perform similarly in semantic robustness. For example, InceptionV3 is more accurate despite being less semantically robust than ResNet50. We hope that this tool will serve as a milestone towards understanding the semantic robustness of DNNs. Figure 4.1 shows how a small perturbation in the view angle of the teapot object results in a drop in InceptionV3 [9] confidence score from 100% to almost 0%. The softmax confidence scores are plotted against one semantic parameter (i.e., the azimuth angle around the teapot) and it fails in such a simple task. Similar
Figure 4.1: **Semantic Robustness of Deep Networks.** Trained neural networks can perform poorly when subject to small perturbations in the semantics of the image. *(left)*: We show how perturbing the azimuth viewing angle of a simple *teapot* object can dramatically affect the score of a pretrained InceptionV3 [9] for the *teapot* class. *(right)*: We plot the softmax confidence scores of different DNNs on the same *teapot* object viewed from 360 degrees around the object. For comparison, lab researchers identified the object from all angles.

behaviors are consistently observed across different DNNs (trained on ImageNet [12]) as noted by other concurrent works [28]. By leveraging a differentiable renderer $R$ and evaluating rendered images for different semantic parameters $u$, we provide a new lens of semantic robustness analysis for such DNNs as illustrated in Figure 4.2. These Network Semantic Maps (NSM) demonstrate unexpected behavior of some DNNs, in which adversarial regions lie inside a very confident region of the semantic space. These regions constitute “traps” that is hard to detect without such analysis and can lead to catastrophic failure for the DNN.
Figure 4.2: **Analysis Pipeline**: We leverage neural mesh renderer $R$ [10] to render shape $S_z$ of class $z$ according to semantic scene parameters $u$. The resulting image is passed to trained network $C$ that is able to identify the class $z$. The behaviour of the softmax score at label $z$ (dubbed $f_z(u)$) is analyzed for different parameters $u$ and for the specific shape $S_z$. In our experiments, we pick $u_1$ and $u_2$ to be the azimuth and elevation angles, respectively.

### 4.1 Related Work

#### 4.1.1 Understanding Deep Neural Networks

There are different lenses to analyze DNNs depending on the purpose of analysis. A popular line of work tries to visualize the network hidden layers by inverting the activations to get a visual image that represents a specific activation [102, 103, 104]. Others observe the behavior of these networks under injected noise [105, 106, 107, 108, 109, 110]. Geirhos *et al.* show that changing the texture of the object while keeping the borders can hugely deteriorate the recognizability of the object by the DNN [111]. More closely related to our work is the work of Fawzi *et al.*, which shows that geometric changes in the image greatly affect the performance of the classifier [112]. The work of Engstrom *et al.* [113] studies the robustness of a network under natural 2D transformations (*i.e.* translation and planar rotation).
4.1.2 Semantic Adversarial Attacks on Deep Neural Networks

The way that DNNs fail for some noise added to the image motivated the adversarial attacks literature. Several works formulate attacking neural networks as an optimization on the input pixels \([21, 22, 23, 24, 25]\). Other works tried to move away from pixel perturbation to semantic 3D scene parameters and 3D attacks \([29, 28, 36, 101]\). Zeng et al. \([29]\) generate attacks on deep classifiers by perturbing scene parameters like lighting and surface normals. Alcorn et al. \([28]\) tries to fool trained DNNs by changing the pose of the object. They used the Neural Mesh renderer (NMR) by Kato et al. \([114]\) to allow for a fully differentiable pipeline that performs adversarial attacks based on the gradients to the parameters. Our work differs in that we use NMR to obtain gradients to the parameters \(u\) not to attack the model, but to detect and quantify the robustness of different networks as shown in Section 4.3.4. Furthermore, Dreossi et al. \([115]\) used adversarial training in the semantic space for self-driving, whereas Liu et al. \([36]\) proposed a differentiable renderer to perform parametric attacks and the \(parametric-ball\) as an evaluation metric for physical attacks. The work by Shu et al. \([116]\) used an RL agent and a Bayesian optimizer to asses the DNNs behaviour under good/bad physical parameters for the network. While we share similar insights as \([116]\), we try to study the global behaviour of DNNs as collections of regions, whereas \([116]\) tries to find individual points that pose difficulty for the DNN.

4.1.3 Optimizing Integral Bounds

**Naive Approach.** To develop an algorithm for robust region finding, we adopt an idea from weakly supervised activity detection in videos by Shou et al. \([117]\). The idea is to find bounds that maximize the inner average of a continuous function while minimizing the outer average in a region. This is achieved because optimizing the bounds to exclusively maximize the area can lead to diverging bounds of \(\{-\infty, \infty\}\).
Figure 4.3: **Semantic Robust Region Finding:** We find robust regions of semantic parameters for ResNet50 [11] and for a *bathtub* object by the three bottom-up formulations (naive, OIR_W, and OIR_B). *(left):* Semantic space is 1D (azimuth angle of camera) with three initial points. *(right):* Semantic space is 2D (azimuth angle and elevation angle of camera) with four initial points.

To solve the issue of diverging bounds, the following naive formulation is to simply regularize the loss by adding a penalty on the region size. The expressions for the loss of $n=1$ dimension is:

$$L = -\text{Area}_{in} + \frac{\lambda}{2} |b - a|_2^2 = \int_a^b f(u) du + \frac{\lambda}{2} |b - a|_2^2,$$

where $f : \mathbb{R}^1 \to (0, 1)$ is the function of interest and $(a, b)$ are the left and right bounds respectively and $\lambda$ is a hyperparameter. The update directions to minimize the loss are:

$$\frac{\partial L}{\partial a} = f(a) - \lambda (b - a); \quad \frac{\partial L}{\partial b} = -f(b) + \lambda (b - a).$$

The regularizer will prevent the region from growing to $\infty$ and the best bounds will be found if the loss is minimized with gradient descent or any similar approach.

**Trapezoidal Approximation.** To extend the naive approach to $n$-dimensions, we face more integrals in the update directions (hard to compute). Therefore, we deploy the following first-order trapezoid approximation of definite integrals. The Newton-Cortes formula for numerical integration [118] states that:

$$\int_a^b f(u) du \approx (b - a) \frac{f(a) + f(b)}{2}.$$

An asymptotic error estimate is given by:

$$\frac{(b-a)^2}{48} [f'(b) - f'(a)] + \mathcal{O}\left(\frac{1}{b-a}\right).$$

So, as long as the derivatives are bounded by some Lipschitz constant $\mathbb{L}$, then the error becomes bounded such that $|\text{error}| \leq \mathbb{L}(b-a)^2$. 
Figure 4.4: **Network Semantic Maps:** We plot the 2D semantic maps of four different networks on two shapes of a *chair* class (*top*) and *cup* class (*bottom*). InceptionV3 is very confident about its decision, but at the cost of creating semantic “traps”, where sharp performance degradation happens in the middle of a robust region. Color maps follow Figure 4.3 right color map.

### 4.2 Methodology

Typical adversarial pixel attacks involve a neural network \( C \) (*e.g.* classifier or detector) that takes an image \( \mathbf{x} \in [0, 1]^d \) as input and outputs a multinomial distribution over \( K \) class labels with softmax values \([l_1, l_2, ..., l_K]\), where \( l_j \) is the softmax value for class \( j \). The adversary (attacker) tries to produce a perturbed image \( \mathbf{x}' \in [0, 1]^d \) that is as close as possible to \( \mathbf{x} \), such that \( \mathbf{x} \) to \( \mathbf{x}' \) have different class predictions through \( C \).

In this work, we consider a more general case where we are interested in parameters \( \mathbf{u} \in \Omega \subset \mathbb{R}^n \), a latent parameter that generates the image via a scene generator (*e.g.* a renderer function \( R \)). This generator/renderer takes the parameter \( \mathbf{u} \) and an object shape \( \mathbf{S} \) of a class that is identified by \( C \). \( \Omega \) is the continuous semantic space for the parameters that we intend to study. The renderer creates the image \( \mathbf{x} \in \mathbb{R}^d \), and then we study the behavior of a classifier \( C \) of that image across multiple shapes and multiple popular DNN architectures. Now, this function of interest is defined as follows:

\[
f(\mathbf{u}) = C_2(R(\mathbf{S}_z, \mathbf{u})) \quad 0 \leq f(\mathbf{u}) \leq 1 \quad (4.1)
\]
where \( z \) is a class label of interest to study and we observe the network score for that class by rendering a shape \( S_z \) of the same class. The shape and class labels are constants and only the parameters \( u \) vary for \( f \) during analysis.

### 4.2.1 Region Finding as an Operator

We can visualize the function in Eq (4.1) for any shape \( S_z \) as long as the DNN can identify the shape at some region in the semantic space \( \Omega \) of interest, as we show in Figure 4.1. However, plotting these figures is expensive and the complexity of plotting them increases exponentially with a big base. The complexity of plotting these plots of semantic maps, which we call Network Semantic Maps (NSM), is \( N \) for \( n = 1 \), where \( N \) is the number of samples needed for that dimension to be fully characterized. The complexity is \( N^2 \) for \( n = 2 \), and we can see that for a general dimension \( n \), the complexity of plotting the NMS to adequately fill the semantic space \( \Omega \) is \( N^n \). This number is intractable even if we have only moderate dimensionality. To tackle this issue, we use a bottom-up approach to detect regions around some initial parameters \( u_0 \), instead of sampling in the entire space of parameters \( \Omega \). Explicitly, we define region finding as an operator \( \Phi \) that takes the function of interest in Eq (4.1), initial point in the semantic space \( u_0 \in \Omega \), and a shape \( S_z \) of some class \( z \). The operator will return the hyper-rectangle \( \mathbb{D} \subset \Omega \), where the DNN is robust in the region and does not sharply drop the score of the intended class. It also keeps identifying the shape with label \( z \) as illustrated in Figure 4.4. The robust-region-finding operator is then defined as follows:

\[
\Phi_{\text{robust}}(f(u), S_z, u_0) = \mathbb{D} = \{ u : a \leq u \leq b \}
\]

\[
\text{s.t. } E_{u \sim \mathbb{D}}[f(u)] \geq 1 - \epsilon_m, \quad u_0 \in \mathbb{D}, \quad \text{VAR}[f(u)] \leq \epsilon_v
\]

where the left and right bounds of \( \mathbb{D} \) are \( a = [a_1, a_2, ..., a_n] \) and \( b = [b_1, b_2, ..., b_n] \), respectively. The two small thresholds \((\epsilon_m, \epsilon_v)\) are needed to ensure high performance
and low variance of the DNN in that robust region. We can define the complementary operator, which finds adversarial regions as:

\[
\Phi_{\text{adv}}(f(u), S_z, u_0) = \mathcal{D} = \left\{ u : a \leq u \leq b \right\}
\]

\[
\text{s.t. } \mathbb{E}_{u \sim \mathcal{D}}[f(u)] \leq \epsilon_m, \quad u_0 \in \mathcal{D}, \quad \text{VAR}[f(u)] \geq \epsilon_v \quad (4.3)
\]

We can clearly show that \( \Phi_{\text{adv}} \) and \( \Phi_{\text{robust}} \) are related:

\[
\Phi_{\text{adv}}(f(u), S_z, u_0) = \Phi_{\text{robust}}(1 - f(u), S_z, u_0) \quad (4.4)
\]

So, we can just focus our attention on \( \Phi_{\text{robust}} \) to find robust regions, and the adversarial regions follow directly from Eq (4.4). We need to ensure that \( \mathcal{D} \) has a positive size: \( r = b - a > 0 \). The volume of \( \mathcal{D} \) normalized by the exponent of dimension \( n \) is expressed as follows:

\[
\text{volume}(\mathcal{D}) = \triangle = \frac{1}{2^n} \prod_{i=1}^{n} r_i \quad (4.5)
\]

The region \( \mathcal{D} \) can also be defined in terms of the matrix \( D \) of all the corner points \( \{d^i\}_{i=1}^{2^n} \) as follows:

\[
\text{corners}(\mathcal{D}) = D_{n \times 2^n} = [d^1|d^2|..|d^{2^n}] = 1^T a + M^T \odot (1^T r)
\]

\[
M_{n \times 2^n} = [m^0|m^1|..|m^{2^n-1}], \quad \text{where } m^i = \text{binary}_n(i) \quad (4.6)
\]

and \( 1 \) is the all-ones vector of size \( 2^n \), \( \odot \) is the Hadamard (element-wise) product of matrices, and \( M \) is a constant masking matrix defined as the permutation matrix of binary numbers of \( n \) bits that range from 0 to \( 2n - 1 \).
4.2.2 Deriving Update Directions

Extending Naive to $n$-dimensions. We start by defining the function vector $f_{D}$ of all function evaluations at all corner points of $D$.

$$f_{D} = \begin{bmatrix} f(d^{1}), f(d^{2}), \ldots, f(d^{2^{n}}) \end{bmatrix}^{T}, \ d^{i} = D_{i} \quad (4.7)$$

Then, using Trapezoid approximation and Leibniz rule of calculus, the loss expression and the update directions become as follows:

$$L(a, b) = -\int \cdots \int_{D} f(u_{1}, \ldots, u_{n}) \, du_{1} \ldots du_{n} + \frac{\lambda}{2} |r|^{2} \approx -\Delta 1^{T} f_{D} + \frac{\lambda}{2} |r|^{2} \quad (4.8)$$

$$\nabla_{a} L \approx 2\Delta \text{diag}^{-1}(r) Mf_{D} + \lambda r \quad ; \quad \nabla_{b} L \approx -2\Delta \text{diag}^{-1}(r) Mf_{D} - \lambda r$$

We show all the derivations for general $n$ in Appendix C.2.

Outer-Inner Ratio Loss (OIR). We introduce an outer region $(A, B)$ with that contains the small region $(a, b)$. We follow the following assumption to ensure that the outer area is always positive: $A = a - \alpha \frac{b-a}{2}; B = b + \alpha \frac{b-a}{2}$. Here, $\alpha$ is the small boundary factor of the outer area to the inner. We formulate the problem as a ratio of outer over inner areas and we try to make this ratio $(L = \frac{\text{Area}_{out}}{\text{Area}_{in}})$ as close as possible to 0. We utilize the Dinkelbach technique for solving non-linear fractional programming problems [121] to transform $L$ as follows.

$$L = \frac{\text{Area}_{out}}{\text{Area}_{in}} = \text{Area}_{out} - \lambda \text{Area}_{in}$$

$$= \int_{A}^{B} f(a) \, da - \int_{a}^{b} f(a) \, da - \lambda \int_{a}^{b} f(a) \, da \quad (4.9)$$

where $\lambda^{*} = \frac{\text{Area}_{out}^{*}}{\text{Area}_{in}^{*}}$ is the Dinkelbach factor that is the best objective ratio.

Black-Box (OIR_B). Here we set $\lambda = 1$ to simplify the problem. This yields the following expression of the loss $L = \text{Area}_{out} - \text{Area}_{in} = \int_{A}^{B} f(u) \, du - 2 \int_{a}^{b} f(u) \, du$, which is similar to the area contrastive loss in [117]. The update rules would be
\[ \frac{\partial L}{\partial a} = -(1 + \frac{\alpha}{2})f(A) - \frac{\alpha}{2} f(B) + 2f(a) ; \quad \frac{\partial L}{\partial b} = (1 + \frac{\alpha}{2})f(B) + \frac{\alpha}{2} f(A) - 2f(b). \]

To extend to \( n \)-dimensions, we define an outer region \( Q \) that includes the smaller region \( D \) and defined as: \( Q = \{ \mathbf{u} : a - \frac{\alpha}{2} \mathbf{r} \leq \mathbf{u} \leq b + \frac{\alpha}{2} \mathbf{r} \} \), where \( (a, b, \mathbf{r}) \) are defined as before, while \( \alpha \) is defined as the boundary factor of the outer region for all the dimensions.

The inner region \( D \) is defined as in Eq (4.6), while the outer region can be defined in terms of the corner points as follows:

\[ \text{corners}(Q) = Q_{n \times 2^n} = \begin{bmatrix} q^1 & q^2 & \cdots & q^{2^n} \end{bmatrix} \]

\[ Q = 1^T (a - \frac{\alpha}{2} \mathbf{r}) + (1 + \alpha) \mathbf{M}^T \circ (1^T \mathbf{r}) \]

Let \( \mathbf{f}_D \) be a function vector as in Eq (4.7) and \( \mathbf{f}_Q \) be another function vector evaluated at all possible outer corner points:

\[ \mathbf{f}_Q = \begin{bmatrix} f(q^1), f(q^2), \ldots, f(q^{2^n}) \end{bmatrix}^T, \; q^i = Q_{n,i}. \]

Now, the loss and update directions for the \( n \)-dimensional case becomes:

\[ L(a, b) = \int \cdots \int_Q f(u_1, \ldots, u_n) \, du_1 \ldots du_n - 2 \int \cdots \int_D f(u_1, \ldots, u_n) \, du_1 \ldots du_n \approx \Delta \left( (1 + \alpha)^n 1^T \mathbf{f}_Q - 2 1^T \mathbf{f}_D \right) \]

\[ \nabla_a L \approx 2\Delta \text{diag}^{-1}(\mathbf{r}) \left( 2\mathbf{M}_D \mathbf{f}_D - \overline{\mathbf{M}}_Q \mathbf{f}_Q \right) ; \quad \nabla_b L \approx 2\Delta \text{diag}^{-1}(\mathbf{r}) \left( -2\mathbf{M}_D + \overline{\mathbf{M}}_Q \mathbf{f}_Q \right) \]

(4.11)

where \( \text{diag}(.) \) is the diagonal matrix of the vector argument or the diagonal vector of the matrix argument. \( \overline{\mathbf{M}}_Q \) is the outer region scaled mask defined as follows:

\[ \overline{\mathbf{M}}_Q = (1 + \alpha)^{n-1} \left( (1 + \frac{\alpha}{2})\mathbf{M} + \frac{\alpha}{2} \mathbf{M} \right) ; \quad \mathbf{M}_Q = (1 + \alpha)^{n-1} \left( (1 + \frac{\alpha}{2})\mathbf{M} + \frac{\alpha}{2} \mathbf{M} \right) \]

(4.12)

**White-Box OIR (OIR\_W).** Here, we present the white-box formulation of Outer-Inner-Ratio. This requires access to the gradient of the function \( f \) in order to update the current estimates of the bound. As we show in Section 4.3, access to gradients enhance the quality of the detected regions. To derive the formulation, We set \( \lambda = \frac{\alpha}{\beta} \) in Eq (4.9), where \( \alpha \) is the small boundary factor of the outer area and \( \beta \) is the
gradient emphasis factor. Hence, the objective in Eq (4.9) becomes:

\[
\arg \min_{a,b} L = \arg \min_{a,b} \text{Area}_{\text{out}} - \lambda \text{Area}_{\text{in}}
\]

\[
= \arg \min_{a,b} \int_{a}^{b} f(u)du + \int_{\beta}^{B} f(u)du - \frac{\alpha}{\beta} \int_{a}^{b} f(u)du
\]

\[
= \arg \min_{a,b} \frac{\beta}{\alpha} \int_{a}^{b} f(u)du - (1 + \frac{\beta}{\alpha}) \int_{a}^{b} f(u)du
\]

\[
\frac{\partial L}{\partial a} = \frac{\beta}{\alpha} \left( f(a) - f \left( a - \frac{b-a}{2} \right) \right) - \frac{\beta}{2} f \left( b + a \frac{b-a}{2} \right) - \frac{\beta}{2} f \left( a - a \frac{b-a}{2} \right) + f(a)
\]

(4.13)

Now, since \( \lambda^* \) should be small for the optimal objective as \( \lambda \to 0, \alpha \to 0 \) and hence the derivative in Eq (4.13) becomes the following:

\[
\lim_{\alpha \to 0} \frac{\partial L}{\partial a} = \frac{\beta}{2} \left( (b-a)f'(a) + f(b) \right) + \left( 1 - \frac{\beta}{2} \right)f(a)
\]

\[
\lim_{\alpha \to 0} \frac{\partial L}{\partial b} = \frac{\beta}{2} \left( (b-a)f'(b) + f(a) \right) - \left( 1 - \frac{\beta}{2} \right)f(b)
\]

(4.14)

We can see that the update rule for \( a \) and \( b \) depends on the function value and the derivative of \( f \) at the boundaries \( a \) and \( b \) respectively, with \( \beta \) controlling the dependence. If \( \beta \to 0 \), the update directions in Eq (4.14) collapse to the unregularized naive update. To extend to \( n \)-dimensions, we have to define a term that involves the gradient of the function, \( i.e. \) the all-corners gradient matrix \( G_D \).

\[
G_D = \left[ \nabla f(d^1) \mid \nabla f(d^2) \mid ... \mid \nabla f(d^n) \right]^T
\]

(4.15)

Now, the loss and update directions are given as follows.

\[
L(a, b) \approx \frac{(1+\alpha)^n \mathbf{1}^T f_d}{\mathbf{1}^T f_d} - 1
\]

\[
\nabla_a L \approx \Delta \left( \text{diag}^{-1}(r) \mathbf{M}_d f_d + \beta \text{diag} (\mathbf{M} G_D) + \beta s \right)
\]

\[
\nabla_b L \approx \Delta \left( -\text{diag}^{-1}(r) \mathbf{M}_d f_d + \beta \text{diag} (\mathbf{M} G_D) + \beta s \right)
\]

(4.16)
Algorithm 1: Robust $n$-dimensional Region Finding for Black-Box DNNs by Outer-Inner Ratios

**Requires:** Semantic Function of a DNN $f(u)$ in Eq (4.1), initial semantic parameter $u_0$, number of iterations $T$, learning rate $\eta$, object shape $S_z$ of class label $z$, boundary factor $\alpha$, Small $\epsilon$

- Form constant binary matrices $M, \overline{M}, M_Q, \overline{M}_Q, M_D, \overline{M}_D$
- Initialize bounds $a_0 = u_0 - \epsilon 1$, $b_0 = u_0 + \epsilon 1$
- $r_0 = a_0 - b_0$, update region volume $\triangle_0$ as in Eq (4.5)

for $t \leftarrow 1$ to $T$

- Form the all-corners function vectors $f_D, f_Q$ as in Eq (4.7,4.10)
- $\nabla_a L \leftarrow 2\triangle_{t-1} \text{diag}^{-1}(r_{t-1})(2Mf_D - \overline{M}_Q f_Q)$
- $\nabla_b L \leftarrow 2\triangle_{t-1} \text{diag}^{-1}(r_{t-1})(-2Mf_D + M_Q f_Q)$
- Update bounds: $a_t \leftarrow a_{t-1} - \eta \nabla_a L$, $b_t \leftarrow b_{t-1} - \eta \nabla_b L$
- $r_t \leftarrow a_t - b_t$, update region volume $\triangle_t$ as in Eq (4.5)

end

Returns: robust region bounds: $a_T, b_T$

where the mask is the special mask

$$M_D = (\gamma_n M - \beta M) ; \quad M_D = (\gamma_n M - \beta \overline{M}) ; \quad \gamma_n = 2 - \beta(2n-1) \quad (4.17)$$

$s$ is a weighted sum of the gradient from other dimensions ($i \neq k$) contributing to the update direction of dimension $k$, where $k \in \{1, 2, ..., n\}$.

$$s_k = \frac{1}{r_k} \sum_{i=1, i \neq k}^{n} r_i ((M_{i,:} - M_{i,:}) \odot M_{k,:}) G_{i,:}$$

$$s_k = \frac{1}{r_k} \sum_{i=1, i \neq k}^{n} r_i ((M_{i,:} - \overline{M}_{i,:}) \odot \overline{M}_{k,:}) G_{i,:} \quad (4.18)$$

Algorithms 1, and 2 summarize the techniques explained above, which we implement in Section 4.3. The derivation of the $n$-dimensional case of the OIR formulation, as well as other unsuccessful formulations are all included in Appendix C.2.


Algorithm 2: Robust $n$-dimensional Region Finding for White-Box DNNs by Outer-Inner Ratios

**Requires:** Semantic Function of a DNN $f(u)$ in Eq (4.1), initial semantic parameter $u_0$, learning rate $\eta$, object shape $S_z$ of class label $z$, emphasis factor $\beta$, Small $\epsilon$

Form constant binary matrices $M, M, M_D, M_D$

Initialize bounds $a_0 \leftarrow u_0 - \epsilon 1, b_0 \leftarrow u_0 + \epsilon 1$

$r_0 \leftarrow a_0 - b_0$, update region volume $\Delta_0$ as in Eq (4.5)

**for** $t \leftarrow 1$ **to** $T$ **do**

**form** the all-corners function vector $f_D$ as in Eq (4.7)

**form** the all-corners gradients matrix $G_D$ as in Eq (4.15)

**form** the gradient selection vectors $s, s$ as in Eq (4.18)

$\nabla_a L \leftarrow \Delta_{t-1} \left( \text{diag}^{-1}(r_{t-1}) M_D f_D + \beta \text{diag}(M_G + \beta s) \right)$

$\nabla_b L \leftarrow \Delta_{t-1} \left( -\text{diag}^{-1}(r_{t-1}) M_D f_D + \beta \text{diag}(M_G) + \beta s \right)$

update bounds: $a_t \leftarrow a_{t-1} - \eta \nabla_a L, b_t \leftarrow b_{t-1} - \eta \nabla_b L$

$r_t \leftarrow a_t - b_t$, update region volume $\Delta_t$ as in Eq (4.5)

**end**

**Returns:** robust region bounds: $a_T, b_T$.


4.3 Experiments

4.3.1 Setup and Data

In this chapter, we chose the semantic parameters $u$ to be the azimuth rotations of the viewpoint and the elevation angle from the horizontal plane, where the object is always at the center of the rendering. This is common practice in the literature [122]. We use 100 shapes from 10 different classes from ShapeNet [16], the largest dataset for 3D models that are normalized from the semantic lens. We pick these 100 shapes specifically such that: (1) the class label is available in ImageNet [12] and that ImageNet classifiers can identify the exact class, and (2) the selected shapes are identified by the classifiers at some part of the semantic space. To do this, we measured the average score in the space and accepted the shape only if its average Resnet softmax score is 0.1. To render the images, we use a differentiable renderer NMR [114], which allows obtaining the gradient to the semantic input parameters.

The networks of interest are Resnet50 [11], VGG [120], AlexNet [119], and InceptionV3 [9]. We use the official PyTorch implementation for each network [88].
### Table 4.1: Benchmarking popular DNNs in Semantic Robustness

We develop the Semantic Robustness Volume Ratio (SRVR) metric to quantify and compare the semantic robustness of well-known DNNs. We see that semantic robustness does not necessarily depend on the accuracy of the DNN.

<table>
<thead>
<tr>
<th>Deep Networks</th>
<th>SRVR</th>
<th>Top-1 error</th>
<th>Top-5 Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet [119]</td>
<td>8.87%</td>
<td>43.45</td>
<td>20.91</td>
</tr>
<tr>
<td>VGG-11 [120]</td>
<td>9.72%</td>
<td>30.98</td>
<td>11.37</td>
</tr>
<tr>
<td>Inceptionv3 [9]</td>
<td>7.92%</td>
<td><strong>22.55</strong></td>
<td><strong>6.44</strong></td>
</tr>
</tbody>
</table>

4.3.2 Mapping the Networks

Similar to Figure 4.1, we map the networks for all 100 shapes on the first semantic parameter (the azimuth rotation), as well as the joint (azimuth and elevation). We show these results in Figure 4.4. The ranges for the two parameters were $[0^\circ, 360^\circ], [-10^\circ, 90^\circ]$, with a $3 \times 3$ grid. The total number of network evaluations is 4K forward passes from each network for every shape (total of 1.6M forward passes). We show all of the remaining results in Appendix C.1.

4.3.3 Growing Semantic Robust Regions

We implement the three bottom-up approaches in Table 4.2 and Algorithms 1 and 2. The hyper-parameters were set to $\eta = 0.1, \alpha = 0.05, \beta = 0.0009, \lambda = 0.1, T = 800$. We can observe in Figure 4.3 that multiple initial points inside the same robust region converge to the same boundary. One key difference to be noted between the naive approach in Eq (4.8) and the OIR formulations in Eq (4.11,4.16) is that the naive approach fails to capture robust regions in some scenarios and fall for trivial regions (see Figure 4.3).

4.3.4 Applications

Quantifying Semantic Robustness.

Looking at these NSM can lead to insights about the network, but we would like to
Table 4.2: Semantic Analysis Techniques: We compare different approaches to analyse the semantic robustness of DNNs.

<table>
<thead>
<tr>
<th>Analysis Approach</th>
<th>Paradigm</th>
<th>Total Sampling complexity</th>
<th>Black-box Functions</th>
<th>Forward pass /step</th>
<th>Backward pass /step</th>
<th>Identification Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Sampling</td>
<td>top-down</td>
<td>$\mathcal{O}(N^n)$ $N \gg 2$</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>Fully identifies the semantic map of DNN</td>
</tr>
<tr>
<td>Naïve</td>
<td>bottom-up</td>
<td>$\mathcal{O}(2^n)$</td>
<td>✓</td>
<td>$2^n$</td>
<td>0</td>
<td>finds strong robust regions only around $u_0$</td>
</tr>
<tr>
<td>OIR_B</td>
<td>bottom-up</td>
<td>$\mathcal{O}(2^{n+1})$</td>
<td>✓</td>
<td>$2^{n+1}$</td>
<td>0</td>
<td>finds strong and weak robust regions around $u_0$</td>
</tr>
<tr>
<td>OIR_W</td>
<td>bottom-up</td>
<td>$\mathcal{O}(2^n)$</td>
<td>X</td>
<td>$2^n$</td>
<td>$2^n$</td>
<td>finds strong and weak robust regions around $u_0$</td>
</tr>
</tbody>
</table>

[Table 4.2: Semantic Analysis Techniques continued...]

**Table 4.2: Semantic Analysis Techniques**: We compare different approaches to analyse the semantic robustness of DNNs.

Develop a systemic approach to quantify the robustness of these DNNs. To do this, we develop the Semantic Robustness Volume Ratio (SRVR) metric. The SRVR of a network is the ratio between the expected size of the robust region obtained by Algorithms 1 and 2 over the nominal total volume of the semantic map of interest. Explicitly, the SRVR of network $C$ for class label $z$ is defined as follows:

$$SRVR_z = \frac{\mathbb{E}[\text{Vol}(\mathcal{D})]}{\text{Vol}(\Omega)} = \frac{\mathbb{E}_{u_0 \sim \Omega, S_z \sim S_z}[\text{Vol}(\Phi(f, S_z, u_0))]}{\text{Vol}(\Omega)}, \quad (4.19)$$

where $f, \Phi$ are defined in Eq (4.1,4.2) respectively. We take the average volume of all the adversarial regions found for multiple initializations and multiple shapes of the same class $z$. Then, we divide by the volume of the entire space. This provides a percentage of how close the DNN is from the ideal behaviour of identifying the object robustly in the entire space. The SRVR metric is not strict in its value, since the analyzer defines the semantic space of interest and the shapes used. However, comparing SRVR scores among DNNs is of extreme importance, as this relative analysis conveys insights about a network that might not be evident by only observing the accuracy of this network. For example, we can see in Table 4.1 that while InceptionV3 [9] is the best in terms of accuracy, it lags behind Resnet50 [11] in terms of semantic robustness. This observation is also consistent with the qualitative NSMs...
Figure 4.5: **Semantic Bias in ImageNet.** By taking the average semantic maps over 10 shapes of *cup* class and over different networks, we can visualize the bias of the training data. The angles of low score are probably not well-represented in ImageNet [12].

in Figure 4.4, in which we can see that while Inception is very confident, it can fail completely inside these confident regions. Note that the reported SRVR results are averaged over all 10 classes and over all 100 shapes. We use 4 constant initial points for all experiments and the semantic parameters are the azimuth and elevation as in Figure 4.3 and 4.4. As can be seen in Figure 4.3, different methods predict different regions, so we take the average size of the the three methods used (naive, OIR_W, and OIR_B) to give an overall estimate of the volume used in the SRVR results reported in Table 4.1.

**Finding Semantic Bias in the Data.**

While observing the above figures uncovers interesting insights about the DNNs and the training data of ImageNet [12], it does not allow to make a conclusion about the network nor about the data. Therefore, we can average these semantic maps of these networks to factor out the effect of the network structure and training and maintain only the effect of training data. We show two such maps called Data Semantic Maps (DSMs). We observe that networks have holes in their semantic maps and these
holes are shared among DNNs, indicating bias in the data. Identifying this gap in the data can help train a more semantically robust network. This can be done by leveraging adversarial training on these data-based adversarial regions as performed in the adversarial attack literature [22]. Figure 4.5 shows an example of this semantic map, which points to the possibility that ImageNet [12] may not contain angles of the easy cup class.

4.4 Analysis

First, we observe in Figure 4.4 that some red areas (adversarial regions the DNN cannot identify the object within) are surrounded by blue areas (robust regions the DNN can identify the object within). These “semantic traps” are dangerous in ML, since they are hard to identify (without NSM) and they can cause failure cases for some models. These traps can be attributed to either the model architecture, training, and loss, or bias in the dataset, on which the model was trained (i.e. ImageNet [12]).

Note that plotting NMS is extremely expensive even for a moderate dimensionality e.g. $n = 8$. For example, for the plot in Figure 4.1, we use $N = 180$ points in the range of 360 degrees. If all the other dimensions require the same number of samples for their individual range, the total joint space requires $180^n = 180^8 = 1.1 \times 10^{18}$ samples, which is enormous. Evaluating the DNN for that many forward passes is intractable. Thus, we follow a bottom-up approach instead, where we start from one point in the semantic space $u_0$ and we grow an $n$-dimensional hyper-rectangle around that point to find the robust “neighborhood” of that point for this specific DNN. Table 4.2 compares different analysis approaches for semantic robustness of DNNs.
Chapter 5

SADA: Semantic Adversarial Diagnostic Attacks for Autonomous Applications

One major factor impeding more widespread adoption of DNNs is their lack of robustness, which is essential for safety-critical applications such as autonomous driving. This has motivated much recent work on adversarial attacks for DNNs, which mostly focus on pixel-level perturbations void of semantic meaning. In this chapter, we present a general framework for adversarial attacks on trained agents, which covers semantic perturbations to the environment of the agent performing the task as well as pixel-level attacks. To do this, we re-frame the adversarial attack problem as learning a distribution of parameters that always fools the agent. In the semantic case, our proposed adversary (denoted as BBGAN) is trained to sample parameters that describe the environment with which the black-box agent interacts, such that the agent performs its dedicated task poorly in this environment. We apply BBGAN on three different tasks, primarily targeting aspects of autonomous navigation: object detection, self-driving, and autonomous UAV racing. On these tasks, BBGAN can generate failure cases that consistently fool a trained agent. Figure 5.1 shows an example of an object misclassified by the YOLOV3 detector [123] applied to a rendered image from a virtual environment, an autonomous UAV racing [124] failure case in a recently developed general purpose simulator (Sim4CV [15]), and an autonomous driving failure case in a popular driving simulator (CARLA [14]). These failures arise from adversarial attacks on the semantic parameters of the environment.
Figure 5.1: **Semantic Adversarial Diagnostic Attacks.** Neural networks can perform poorly or downright fail when encountering some environments. To diagnose why they fail and how they can be improved, we seek to learn the underlying distribution of semantic parameters, which generate environments that pose difficulty to these networks when applied to three safety critical tasks.

5.1 Related Work

5.1.1 Semantic Attacks and Simulations

Beyond pixel perturbations, several recent works perform attacks on the object/camera pose to fool a classifier [28, 125, 126]. Other works proposed attacks on 3D point clouds using Point-Net [127], and on 3D meshes using differentiable functions that describe the scene [128]. Inspired by these excellent works, we extend semantic attacks by using readily available virtual environments with plausible 3D setups to systematically test trained agents. In fact, our formulation includes attacks not only on static agents like object detectors, but also agents that interact with dynamic environments, such as self-driving agents. To the best of our knowledge, this is the first work to introduce adversarial attacks in CARLA[14], a standard autonomous navigation benchmark.
Figure 5.2: **Generic Adversarial Attacks on Agents.** $E_u$ is a parametric environment with which an agent $A$ interacts. The agent receives an observation $o_t$ from the environment and produces an action $a_t$. The environment scores the agent and updates its state until the episode finishes. A final score $Q(A, E_u)$ is given to the adversary $G$, which in turn updates itself to propose more adversarial parameters $u$ for the next episode.

### 5.1.2 Adversarial Attacks and Reinforcement Learning

Our generic formulation of adversarial attacks is naturally inspired by RL, in which agents can choose from multiple actions and receive partial rewards as they proceed in their task [129]. In RL, the agent is subject to training in order to achieve a goal in the environment; the environment can be dynamic to train a more robust agent [130, 131, 132]. However, in adversarial attacks, the agent is usually fixed and the adversary is the subject of the optimization in order to fool the agent. We formulate adversarial attacks in a general setup where the environment rewards an agent for some task. An adversary outside the environment is tasked to fool the agent by modifying the environment and receiving a score after each episode.
5.2 Methodology

5.2.1 Generalizing Adversarial Attacks

Extending attacks to general agents. In this work, we generalize the adversarial attack setup beyond pixel perturbations. Our more general setup (refer to Figure 5.2) includes semantic attacks, e.g., perturbing the camera pose or lighting conditions of the environment that generates observations (e.g., pixels in 2D images). An environment $E_u$ is parametrized by $u \in [u_{\text{min}}, u_{\text{max}}]^d$. It has an internal state $s_t$ and produces observations $o_t \in \mathbb{R}^n$ at each time step $t \in \{1, \ldots, T\}$. The environment interacts with a trained agent $A$, which gets $o_t$ from $E_u$ and produces actions $a_t$. At each time step $t$ and after the agent performs $a_t$, the internal state of the environment is updated: $s_{t+1} = E_u(s_t, a_t)$. The environment rewards the agent with $r_t = R(s_t, a_t)$, for some reward function $R$. We define the episode score $Q(A, E_u) = \sum_{t=1}^{T} r_t$ of all intermediate rewards. The goal of $A$ is to complete a task by maximizing $Q$.

The adversary $G$ attacks the agent $A$ by modifying the environment $E_u$ through its parameters $u$, without access to $A$ and $E_u$.

Distribution of Adversarial Attacks. We define $P_{u'}$ to be the fooling distribution of semantic parameters $u'$ representing the environments $E_{u'}$, which fool the agent $A$.

$$u' \sim P_{u'} \Leftrightarrow Q(A, E_{u'}) \leq \epsilon; \quad u' \in [u_{\text{min}}, u_{\text{max}}]^d \quad (5.1)$$

Here, $\epsilon$ is a task-specific threshold to determine success and failure of the agent $A$. The distribution $P_{u'}$ covers all samples that result in failure of $A$. Its PDF is unstructured and depends on the complexity of the agent. We seek an adversary $G$ that learns $P_{u'}$, so it can be used to comprehensively analyze the weaknesses of $A$. Unlike the common practice of finding adversarial examples (e.g., individual images), we address the attacks distribution-wise in a compact semantic parameter space. We denote our analysis technique as Semantic Adversarial Diagnostic Attack (SADA).
Algorithm 3: Generic Adversarial Attacks on Agents

Returns: Attack Success Rate (ASR)

Requires: Agent $A$, Adversary $G$, Environment $E_u$, number of episodes $T$, training iterations $L$, test size $M$, fooling threshold $\epsilon$

Training $G$: for $i \leftarrow 1$ to $L$ do
  Sample $u_i \sim G$ and initialize $E_{u_i}$ with initial state $s_1$
  for $t \leftarrow 1$ to $T$ do
    $E_{u_i}$ produces observation $o_t$ from $s_t$
    $A$ performs $a_t(o_t)$ and receives $r_t \leftarrow R(s_t, a_t)$
    State updates: $s_{t+1} \leftarrow E_{u_i}(s_t, a_t)$
  end
  $G$ receives the episode score $Q_i(A, E_{u_i}) \leftarrow \sum_{t=1}^{T} r_t$
  Update $G$ to solve for Eq (5.2)
end

Testing $G$: Initialize fooling counter $f \leftarrow 0$
for $j \leftarrow 1$ to $M$ do
  sample $u_j \sim G$ and initialize $E_{u_j}$ with initial state $s_1$
  for $t \leftarrow 1$ to $T$ do
    $a_t(o_t) ; r_t \leftarrow R(s_t, a_t) ; s_{t+1} \leftarrow E_{u_j}(s_t, a_t)$
  end
  $Q_j(A, E_{u_j}) \leftarrow \sum_{t=1}^{T} r_t$
  if $Q_j(A, E_{u_j}) \leq \epsilon$ then
    $f \leftarrow f + 1$
  end
end

Returns: $\text{ASR} = f/M$

Semantic because of the nature of the environment parameters and diagnostic because a fooling distribution is sought. We show later how this distribution can be used to reveal agents’ failure modes. We propose to optimize the following objective for the adversary $G$ to achieve this challenging goal:

$$\arg \min_{G} \mathbb{E}_{u \sim G}[Q(A, E_u)]$$

s.t. $\{u : u \sim G\} = \{u' : u' \sim P_u\}$

(5.2)

Algorithm 3 describes a general setup for $G$ to learn to generate fooling parameters. It also includes a mechanism for evaluating $G$ in the black-box environment $E_u$ for $L$ iterations after training it to attack the agent $A$. An attack is considered a fooling attack, if parameter $u$ sampled from $G$ achieves an episode score $Q(A, E_u) \leq \epsilon$. 
Figure 5.3: **BBGAN: Learning Fooling Distribution of Semantic Environment Parameters.** We learn an adversary $G_u$, which samples semantic parameters $u$ that parametrize the environment $E_u$, such that an agent $A$ fails in a given task in $E_u$. The inducer produces the induced set $S_u'$ from a uniformly sampled set $\Omega$ by filtering the lowest scoring $u$ (according to $Q$ value), and passing $S_u'$ for BBGAN training. Note that $Q_1 \leq Q_s \ldots \leq Q_N$, where $s = |S_u'|$, $N = |\Omega|$. The inducer and the discriminator are only used during training (dashed lines), after which the adversary learns the fooling distribution $P_{u'}$. Three safety-critical applications are used to demonstrate this in three virtual environments: object detection (in Blender [13]), self-driving cars (in CARLA [14]), and autonomous racing UAVs (in Sim4CV [15]).

Consequently, the Attack Success Rate (ASR) is defined as the rate at which samples from $G$ are fooling attacks. In addition to ASR, the algorithm returns the set $S_u'$ of adversarial examples that can be used to diagnose the agent. The equality constraint in Eq (5.2) is very strict to include all fooling parameters $u'$ of the fooling distribution. It acts as a perceptuality metric in our generalized attack to prevent unrealistic attacks. Next, we relax this equality constraint to leverage recent advances in GANs for learning an estimate of the distribution $P_{u'}$.

### 5.2.2 Black-Box Generative Adversarial Network

Generative Adversarial Networks (GANs) are a promising family of unsupervised techniques that can model complex domains, e.g. natural images [38, 39, 133]. GANs consist of a discriminator $D_x$ and a generator $G_x$ that are adversarially trained to optimize the loss $L_{GAN}(G_x, D_x, P_X)$, where $P_X$ is the distribution of images in
domain X and \( z \in \mathbb{R}^c \) is a latent random Gaussian vector.

\[
\min_{G_x} \max_{D_x} \ L_{\text{GAN}}(G_x, D_x, P_x) = \\
\mathbb{E}_{x \sim p_x(x)}[\log D_x(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D_x(G_x(z)))]
\]

\( D_x \) tries to determine if a given sample (e.g., image \( x \)) is real (exists in the training dataset) or fake (generated by \( G_x \)). On the other hand, \( G_x \) tries to generate samples that fool \( D_x \) (e.g., misclassification). Both networks are proven to converge when \( G_x \) can reliably produce the underlying distribution of the real samples [38].

We propose to learn the fooling distribution \( P_{u'} \) using a GAN setup, which we denote as black-box GAN (BBGAN). We follow a similar GAN objective but replace the image domain \( x \) by the semantic environment parameter \( u \). However, since we do not have direct access to \( P_{u'} \), we propose a module called the \emph{inducer}, which is tasked to produce the induced set \( S_{u'} \) that belongs to \( P_{u'} \). In essence, the \emph{inducer} tries to choose a parameter set which represents the fooling distribution to be learnt by the BBGAN as well as possible. In practice, the inducer selects the best fooling parameters (based on the \( Q \) scores of the agent under these parameters) from a set of randomly sampled parameters in order to construct this induced set. Thus, this setup relaxes Eq (5.2) to:

\[
\arg \min_{G} \ E_{u \sim G}[Q(A, E_u)] \\
\text{s.t.} \ \{ u : u \sim G \} \subset \{ u' : u' \sim P_{u'} \}
\]

So, the final BBGAN loss becomes:

\[
\min_{G_u} \max_{D_u} \ L_{\text{BBGAN}}(G_u, D_u, S_{u'}) = \\
\mathbb{E}_{u \sim S_{u'}}[\log D_u(u)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]
\]

\( G_u \) and \( D_u \) are the networks that generate and discriminate between real and fake samples, respectively. The inducer \( S_{u'} \) determines the induced set of parameters that best fool the discriminative network \( D_u \).
Here, $G_u$ is the generator acting as the adversary, and $z \in \mathbb{R}^m$ is a random variable sampled from a normal distribution. A simple inducer can be just a filter that takes a uniformly sampled set $\Omega = \{u_i \sim \text{Uni}([u_{\min}, u_{\max}])\}^{i=N}_{i=1}$ and suggests the lowest $Q$-scoring $u_i$ that satisfies the condition $Q(u_i) \leq \epsilon$. The selected samples constitute the induced set $S_{u'}$. The BBGAN treats the induced set as a training set, so the samples in $S_{u'}$ act as virtual samples from the fooling distribution $P_{u'}$ that we want to learn. As the induced set size $S_{u'}$ increases, the BBGAN learns more of $P_{u'}$. As $|S_{u'}| \to \infty$, any sample from $S_{u'}$ is a sample of $P_{u'}$ and the BBGAN in Eq (5.5) satisfies the strict condition in Eq (5.2). Consequently, sampling from $G_u$ would consistently fool agent $A$. We show an empirical proof for this in Appendix D.1 and show how we consistently fool three different agents by samples from $G_u$ in the experiments. The number of samples needed for $S_{u'}$ to be representative of $P_{u'}$ depends on the dimensionality $d$ of $u$. Because of the black-box and stochastic nature of $E_u$ and $A$ (similar to other RL environments), we follow the random sampling scheme common in RL [134] instead of deterministic gradient estimation. In the experiments, we compare our method against baselines that use different approaches to solve Eq (5.2).

5.2.3 Special Cases of Adversarial Attacks

One can show that the generic adversarial attack framework detailed above includes well-known types of attacks as special cases, as summarized in Appendix D.2. In fact, the general setup allows for static agents (e.g., classifiers and detectors) as well as dynamic agents (e.g., an autonomous agent acting in a dynamic environment). It also covers pixel-wise image perturbations, as well as, semantic attacks that try to fool the agent in a more realistic scenario. The generic attack also allows for a more flexible way to define the attack success based on an application-specific threshold $\epsilon$ and the agent score $Q$. 

Figure 5.4: **Object Detection Attack Setup**: (Left): the 100 shapes from Pascal3D and ShapeNet of 12 object classes, used to uncover the failure cases of the YOLOV3 detector. (Right): the semantic parameters $u$ defining the environment.

### 5.3 Applications

#### 5.3.1 Object Detection

Object detection is one of the core perception tasks commonly used in autonomous navigation. Based on its suitability for autonomous applications, we choose the very fast, state-of-the-art YOLOv3 object detector [123] as the agent in our SADA framework. We use the open-source software Blender to construct a scene based on freely available 3D scenes and CAD models. We pick an urban scene with an open area to allow for different rendering setups. The scene includes one object of interest as well as a camera and main light source directed toward the center of the object. The light is a fixed strength spotlight located at a fixed distance from the object. The material of each object is semi-metallic, which is common for the classes under consideration. The 3D collection consists of 100 shapes of 12 object classes (aeroplane, bench, bicycle, boat, bottle, bus, car, chair, dining table, motorbike, train, truck) from Pascal-3D [135] and ShapeNet [16]. At each iteration, one shape from the intended class is randomly picked and placed in the middle of the scene. The rendered image is then passed to YOLOV3 for detection. For the environment parameters, we use eight parameters that have shown to affect detection performance and frequently occur in real setups (refer to Figure 5.4).
5.3.2 Self-Driving

There is a lot of recent work in autonomous driving especially in the fields of robotics and computer vision [136, 137]. In general, complete driving systems are very complex and difficult to analyze or simulate. By learning the underlying distribution of failure cases, our work provides a safe way to analyze the robustness of such a complete system. While our analysis is done in simulation only, we would like to highlight that sim-to-real transfer is a very active research field nowadays [138, 139]. We use an autonomous driving agent (based on CIL [137]), which was trained on the environment $E_u$ with default parameters. The driving-policy was trained end-to-end to predict car controls given an input image and is conditioned on high-level commands (e.g. turn right at the next intersection). The environment used is CARLA driving simulator [14], the most realistic open-source urban driving simulator currently available. We consider the three common tasks of driving in a straight line, completing one turn, and navigating between two random points. The score is measured as the average success of five pairs of start and end positions. Since experiments are time-consuming, we restrict ourselves to three parameters, two of which pertain to the mounted camera viewpoint and the third controls the appearance of the environment by changing the weather setting (e.g. ‘clear noon’, ‘clear sunset’, ‘cloudy after rain’, etc.). As such, we construct an environment by randomly perturbing the position and rotation of the default camera along the z-axis and around the pitch axis respectively, and by picking one of the weather conditions. Intuitively, this helps measure the robustness of the driving policy to the camera position (e.g. deploying the same policy in a different vehicle) and to environmental conditions.

5.3.3 UAV Racing

In recent years, UAV (unmanned aerial vehicle) racing has emerged as a new sport where pilots compete in navigating small UAVs through race courses at high speeds.
Since this is a very interesting research problem, it has also been picked up by the robotics and vision communities [140]. We use a fixed agent to autonomously fly through each course and measure its success as percentage of gates passed [141]. If the next gate was not reached within 10 seconds, we reset the agent at the last gate. We also record the time needed to complete the course. The agent uses a perception network that produces waypoints from image input and a PID controller to produce low-level controls. We use the general-purpose simulator for computer vision applications, Sim4CV [15]. Here, we change the geometry of the race course environment rather than its appearance. We define three different race track templates with 3-5 2D anchor points, respectively. These points describe a second order B-spline and are perturbed to generate various race tracks populated by uniformly spaced gates. Please refer to Appendix D.6 for more details.
Table 5.1: **Attack Success Rate (ASR) Comparison:** ASR of adversarial samples generated on three safety-critical applications: YOLOV3 object detection, self-driving, and UAV racing. For detection, we report the average ASR performance across all 12 classes and highlight 3 specific ones. For autonomous driving, we compute the ASR for the three common tasks in CARLA. For UAV racing, we compute ASR for race tracks of varying complexity. Best results are in **bold**.

<table>
<thead>
<tr>
<th></th>
<th>Object Detection</th>
<th>Autonomous Driving</th>
<th>UAV Track Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bike</td>
<td>Motorbike</td>
<td>Truck</td>
</tr>
<tr>
<td>Full Set</td>
<td>14.6</td>
<td>32.5</td>
<td>56.8</td>
</tr>
<tr>
<td>Random</td>
<td>13.3</td>
<td>38.8</td>
<td>73.8</td>
</tr>
<tr>
<td>Multi-Class SVM</td>
<td>20.0</td>
<td>45.6</td>
<td>70.8</td>
</tr>
<tr>
<td>GP Regression</td>
<td>17.6</td>
<td>43.6</td>
<td>83.6</td>
</tr>
<tr>
<td>Gaussian</td>
<td>19.6</td>
<td>40.4</td>
<td>72.4</td>
</tr>
<tr>
<td>GMM10</td>
<td>26.0</td>
<td>48.4</td>
<td>75.2</td>
</tr>
<tr>
<td>GMM50</td>
<td>16.4</td>
<td>46.8</td>
<td>72.0</td>
</tr>
<tr>
<td>Bayesian</td>
<td>48.0</td>
<td>52.0</td>
<td>75.6</td>
</tr>
<tr>
<td>BBGAN (vanilla)</td>
<td>44.0</td>
<td>45.2</td>
<td>90.8</td>
</tr>
<tr>
<td>BBGAN (boost)</td>
<td><strong>65.8</strong></td>
<td><strong>82.0</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>

5.4 Experiments

5.4.1 BBGAN

**Training.** To learn the fooling distribution $P_u'$, we train the BBGAN using a vanilla GAN model [38]. Both, the generator $G$ and the discriminator $D$ consist of a MLP with 2 layers. We train the GAN following convention, but since we do not have access to the true distribution that we want to learn (*i.e.* real samples), we *induce* the set by randomly sampling $N$ parameter vector samples $u$, and then picking the $s$ worst among them (according to the score $Q$). For object detection, we use $N = 20000$ image renderings for each class (a total of 240K images). Due to the computational cost, our dataset for the autonomous navigation tasks comprises only $N = 1000$ samples. For instance, to compute one data point in autonomous driving, we need to run a complete episode that requires 15 minutes. The induced set size is always fixed to be $s = 100$.

**Boosting.** We use a boosting strategy to improve the performance of our BBGAN. Our boosting strategy simply utilizes the samples generated by the previous stage
adversary $G_{k-1}$ in inducing the training set for the current stage adversary $G_k$. This is done by adding the generated samples to $\Omega$ before training $G_k$. The intuition here is that the main computational burden in training the BBGAN is not the GAN training itself, but computing the agent episodes, each of which can take multiple hours for the case of self-driving. For more details, including the algorithm, a mathematical justification and more experimental results please refer to Appendix D.3.

5.4.2 Testing, Evaluation, and Baselines

To highlight the merits of BBGAN, we seek to compare it against baseline methods, which also aim to estimate the fooling distribution $P_{u'}$. In this comparative study, each method produces $M$ fooling/adversarial samples (250 for object detection and 100 for self-driving and UAV racing) based on its estimate of $P_{u'}$. Then, the attack success rate (ASR) for each method is computed as the percentage of the $M$ adversarial samples that fooled the agent. To determine whether the agent is fooled, we use a fooling rate threshold $\epsilon$.

**Gaussian Mixture Model (GMM).** We fit a full covariance GMM of varying Gaussian components to estimate the distribution of the samples in the induced set $S_{u'}$. The variants are denoted as Gaussian (one component), GMM10% and GMM50% (number of components as percentage of the samples in the induced set).

**Bayesian.** We use the Expected Improvement (EI) Bayesian Optimization algorithm [143] to minimize the score $Q$ for the agent. The optimizer runs for $10^4$ steps and it tends to gradually sample more around the global minimum of the function. We use the last $N = 1000$ samples to generate the induced set $S_{u'}$ and then learn a GMM with different Gaussian components. Finally, we sample $M$ parameter vectors from
Figure 5.6: **Visualization of the Fooling Distribution.** (right): We plot the camera positions and light source directions of 250 sampled parameters in a 3D sphere around the object. (left): We show how real photos of a toy car, captured from the same angles as rendered images, confuse the YOLOV3 detector in the same way.

the GMMs and report results for the best model.

**Multi-Class SVM.** We bin the score $Q$ into 5 equally sized bins and train a multiclass SVM classifier on the complete set $Ω$ to predict the correct bin. We then randomly sample parameter vectors $u$, classify them, and sort them by the predicted score. We pick $M$ samples with the lowest $Q$ score.

**Gaussian Process Regression.** Similar to the SVM case, we train a Gaussian Process Regressor [144] with an exponential kernel to regress the scores $Q$ from the corresponding $u$ parameters that generated the environment on the dataset $Ω$.

### 5.4.3 Results

Table 5.1 summarizes the ASR results for the aforementioned baselines and our BBGAN approach across all three applications. For object detection, we show 3 out of 12 classes and report the average across all classes. For autonomous driving, we report the results on all three driving tasks. For UAV racing, we report the results for three different track types, parameterized by an increasing number of 2D anchor
points (3, 4 and 5) representing $\mathbf{u}$. Our results show that we consistently outperform the baselines, even the ones that were trained on the complete set $\Omega$ rather than the smaller induced set $S_u$, such as the multi-class SVM and the GP regressor. While some baselines perform well on the autonomous driving application where $\mathbf{u}$ consists of only 3 parameters, our approach outperforms them by a large margin on the tasks with higher dimensional $\mathbf{u}$ (e.g. object detection and UAV racing with 5-anchor tracks). Our boosting strategy is very effective and improves results even further with diminishing returns for setups where the vanilla BBGAN already achieves a very high or even the maximum success rate.

To detect and prevent mode collapse, a GAN phenomenon where the generator collapses to generate a single point, we do the following. (1) We visualize the Nearest Neighbor (NN) of the generated parameters in the training set as in Figure 5.7. (2) We visualize the distributions of the generated samples and ensure their variety as in Figure 5.6. (3) We measure the average standard deviation per parameter dimension to make sure it is not zero. (4) We visualize the images/tracks created by these parameters as in Figure 5.5.

5.4.4 Analysis

Diagnosis. The usefulness of SADA lies in that it is not only an attacking scheme using BBGAN, but also serves as a diagnosis tool to assess the systematic failures of agents. We perform diagnostic studies (refer to Figure 5.6) to identify cases of systematic failure for the YOLOv3 detector.

Nearest Neighbor Visualization. In Figure 5.7, we visualize the NN in the parameter space for four different generated samples by our BBGAN. We see that the generated and the NN in training are different for the 4 samples with $L_2$ norm differences of $(0.76,0.60,0.81,0.56)$ and $(378,162,99,174)$ in parameter space and in image space respectively (all range from -1 to 1). This shows that our BBGAN can
Figure 5.7: Nearest Neighbor in Training: *(top)*: generated fooling samples by our BBGAN for 4 different classes. *(bottom)*: the corresponding NN from the training set. We can see that our BBGAN generates novel fooling parameters that are not present in training.

5.5 Insights from SADA

**Object Detection with YOLOV3.** For most objects, top-rear or top-front views of the object tend to fool the YOLOV3 detector. The color of the object does not play a significant role in fooling the detector, but usually colors that are closer to the background color tend to be preferred by the BBGAN samples.

**Self-Driving.** Weather is the least important parameter for fooling the driving policy indicating that the policy was trained to be insensitive to this factor. Interestingly, the learned policy is very sensitive to slight perturbations in the camera pose (height and pitch), indicating a systemic weakness that should be ratified with more robust training.

**UAV Autonomous Navigation.** We observe that the UAV fails if the track has very sharp turns. This makes intuitive sense and the results that were produced by our BBGAN consistently produce such tracks. While the tracks that are only
parameterized by three control points can not achieve sharp turns, our BBGAN is
still able to make the UAV agent fail by placing the racing gates very close to each
other, thereby increasing the probability of hitting them.
Chapter 6

Conclusion and Future Work

6.1 Summary

In this dissertation, we addressed the lack of robustness of deep learning models in 3D setups. We start by addressing the robustness of models that operate directly on 3D data (3D point clouds and multi-view images). On 3D point clouds, we define strong adversarial attacks that are transferable between deep models. Using a point cloud autoencoder in crafting the attack generalizes the attack to be hazardous for unseen networks, allowing for black-box transferability, unlike previous works in the literature. On the other hand, multi-view projection methods achieve high performance on 3D shape recognition by learning to aggregate information from multiple views. However, these methods typically use fixed, heuristically-set camera viewpoints. To address this, we propose the Multi-View Transformation Network (MVTN) which regresses optimal viewpoints for 3D shape recognition, using advances in differentiable rendering. This approach improves both the performance of 3D recognition and the robustness to natural noise such as rotation and occlusion.

Next, we addressed another type of attacks on 2D models that are semantically meaningful (i.e. can happen in realistic 3D scenarios). We develop these semantic (physical) attacks on deep image classifiers by perturbing the 3D setup from which the images are generated. We develop a methodology to detect and quantify the regions in the parametric space that pose a challenge to the deep classifiers. We extend these semantic 3D attacks to other applications in computer vision like object detection
and autonomous navigation and show that it is possible to learn a distribution of these hazardous parameters by another class of deep generative models.

6.2 Future Work

Moving forward, we are interested in further developing robust deep learning models for 3D understanding and generation by investigating the following points:

- **Robustness of 3D Segmentation Models.** One of the most widely used applications in 3D computer vision is the ability to segment 3D point clouds of shapes, scenes, and outdoor scans. This task is of primary focus in research with many methods proposed that use deep networks, however, little attention is paid to the robustness of the networks to natural conditions like occlusion, rotation, and view-point perturbations or to crafted adversarial attacks.

- **Propose more diverse and labeled 3D datasets for proper 3D deep learning.** The SPARF [145] dataset we collected is our first attempt to build a large-scale high-quality 3D dataset in radiance field research. However, this is only one of many possible datasets that are missing from the 3D deep learning literature. Scale and level of detail (LoL) are not the only aspects that dictate the possible genres of datasets that can be built in 3D. Other factors include compositions, diversity, interactivity, and animations.

- **Improve Few-shot 3D understanding and generation.** Other than collecting larger 3D datasets, working on improving the data efficiency of 3D learning algorithms is underexplored. Many works address specific tasks by designing specific efficient algorithms for those tasks, which does not transfer to other tasks.

- **Combine 3D data modalities (point clouds, multi-view, voxels, implicit, mesh) for efficient learning.** There is no standard way of representing 3D data, as it
mostly depends on the task or application. Just like how pixels and language tokens facilitated the success of deep learning in 2D vision and NLP respectively, seeking a unified representation in 3D is a valid quest. One of the under-explored areas in this direction is the hybrid representation of multiple 3D modalities on the same task, as we demonstrated successfully in MVTN [146] and Voint Cloud [147].
REFERENCES


shapenets: A deep representation for volumetric shapes,” in Proceedings of
the IEEE conference on computer vision and pattern recognition, 2015, pp.
1912–1920.

multi-view cnns for object classification on 3d data,” in Proceedings of the IEEE
conference on computer vision and pattern recognition, 2016, pp. 5648–5656.

3d object retrieval,” in Proceedings of the IEEE Conference on Computer Vision

convolutional neural network for 3d shape retrieval,” in Proceedings of the

synthesis from a single image,” in Proceedings of the IEEE/CVF Conference
on Computer Vision and Pattern Recognition, 2020, pp. 7467–7477.

[82] M. Sugiyama, “Dimensionality reduction of multimodal labeled data by local
fisher discriminant analysis,” Journal of machine learning research, vol. 8, no.

metrics,” in Proceedings of the 24th annual conference on Computer graphics


of point cloud and multi-view for 3d shape recognition,” in Proceedings of the

Guibas, “Kpconv: Flexible and deformable convolution for point clouds,” in Proceedings of the IEEE/CVF International Conference on Computer Vision,

2020.


APPENDICES
A Appendix A

A.1 Background on Point Cloud Distances

We define a point cloud $\mathcal{X} \in \mathbb{R}^{N \times 3}$, as a set of $N$ 3D points, where each point $x_i \in \mathbb{R}^3$ is represented by its 3D coordinates $(x_i, y_i, z_i)$. In this work, we focus solely on the perturbations of the input. This means we modify each point $x_i$ by a perturbation variable. Formally, we define the perturbed point set $\mathcal{X}' = \mathcal{X} + \Delta$, where $\Delta \in \mathbb{R}^{N \times 3}$ is the perturbation parameter we are optimizing for. Consequently, each pair $(x_i, x'_i)$ are in correspondence.

A.1.1 Trivial Distances ($\ell_p$)

The most commonly used distance metric in adversarial attacks in the image domain is $\ell_p$. Unlike image domain where every pixel corresponds to the perturbed pixel in adversarial attacks, in point clouds adversarial attacks by adding, removing, or transforming the point cloud destroys the correspondence relationship to the unperturbed sample. Hence, it becomes infeasible to accurately calculate the $\ell_p$ metric for the attack. In our paper, we focus on adversarial perturbations, which preserves the matching between the unperturbed sample and the perturbed sample. This property of preservation of matching points allows us to measure the $\ell_p$ norms of the attack exactly, which allow for standard evaluation similar to the one in the image domain. Here we assume the two point-sets are equal in size and are aligned, i.e. for $x_i \in \mathcal{X}$, $x'_i = x_i + \delta_i$, $i \in 1, 2, ..., N$

$$D_{\ell_p} (\mathcal{X}, \mathcal{X}') = \left( \sum_i \| \delta_i \|_p^p \right)^{\frac{1}{p}}$$  \hspace{1cm} (A.1)
For our attacks, we use the $\ell_2$ and $\ell_\infty$ distances, defined in (A.2) and (A.3) respectively. The $\ell_2$ distance measures the energy of the perturbation, while $\ell_\infty$ represents the maximum allowed perturbation of each $\delta_i \in \Delta$.

$\ell_2$ distance.. The $\ell_2$ measures the energy of the perturbation performed on the unperturbed point cloud. Its calculation is similar to calculating the Frobenius norm of the matrix $X$ that represent the point set perturbation variable $\Delta$ such that each row of $X$ is a point $\delta_i \in \Delta$. The $\ell_2$ distance between two point sets can be measured as follows

$$D_{\ell_2}(\mathcal{X}, \mathcal{X}') = \left( \sum_i \|\delta_i\|_2^2 \right)^{\frac{1}{2}} = \|\Delta\|_F \quad (A.2)$$

$\ell_\infty$ distance.. The $\ell_\infty$ represents the max allowed perturbation at any dimension to every single point $\delta_i$ in the perturbation set $\Delta$. This distance between two point sets can be measured as follows:

$$D_{\ell_\infty}(\mathcal{X}, \mathcal{X}') = \max_i \|\delta_i\|_\infty \quad (A.3)$$

### A.1.2 Non-trivial Distances

Other point cloud distances that are commonly used in the literature do not require the two sets to be in a known correspondence (like the strict $\ell_p$). These distance metrics include the following: Chamfer Distances, Hausdorff Distance, and Earth Mover Distance. In what follows, we formally present each of these metrics.

**Chamfer Distance (CD).** This is a common distance to compare 2 point sets. CD measures the average distance between closest point pairs of 2 different point clouds. We define CD in Eq (A.4).

$$D_{CD}(\mathcal{X}, \mathcal{X}') = \frac{1}{\|\mathcal{X}'\|_0} \sum_{x_i' \in \mathcal{X}'} \min_{x_i \in \mathcal{X}} \|x_i - x_i'\|_2^2 \quad (A.4)$$

**Hausdorff distance (HD).** With HD, we compute the largest distance in the set
Earth Mover Distance (EMD). The EMD measures the total effort performed in the optimal transport scheme that transforms the first point set to the other. It is defined as follows:

\[ D_{EMD}(\mathcal{X}, \mathcal{X}') = \min_{\phi: \mathcal{X} \rightarrow \mathcal{X}'} \sum_i \| x_i' - \phi(x_i) \|_2, \]  

(A.6)

where \( \phi: \mathcal{X} \rightarrow \mathcal{X}' \) is a bijection transform.

A.2 Full Transferability Results

A.2.1 Transferability on Different Norms

In Figure A.1 the attacks are optimized using different \( \epsilon_\infty \) norm-budgets. We report the attack success on all victim networks and the success of these attacks on each transfer network. We note that our AdvPC transfers better to the other networks across different \( \epsilon_\infty \) as compared to the baselines 3D-adv[4] and KNN attack [5].

A.3 Defenses Results

A.4 Analysis of the results

A.4.1 Ablation Study (hyperparameter \( \gamma \))

Here, we study the effect of the transferability hyper parameter \( \gamma \) on the performance of our attacks. While varying \( \gamma \) between 0 and 1, we record the attack success rate on the victim network and report the transferability to all of the other three transfer networks (average success rate on the transfer networks). We present our results
Figure A.1: Transferability Across Different $\epsilon_\infty$ Norm-Budgets: Here, the attacks are optimized using different $\epsilon_\infty$ norm-budgets. We report the attack success on all victim networks and the success of these attacks on each transfer network.
Table A.1: Attacking Point Cloud Defenses (DGCNN): We evaluate untargeted attacks using norm-budgets of $\epsilon_{\infty} = 0.18$ and $\epsilon_{\infty} = 0.45$ with DGCNN [3] as the victim network under different defenses for 3D point clouds. Similar to before, we report attack success rates (higher indicates better attack). Note that both the attacks and evaluations are performed on DGCNN, which has an accuracy of 93.7% without input perturbations (for reference).

<table>
<thead>
<tr>
<th>Defenses</th>
<th>$\epsilon_{\infty} = 0.18$</th>
<th>$\epsilon_{\infty} = 0.45$</th>
</tr>
</thead>
<tbody>
<tr>
<td>No defense</td>
<td>100</td>
<td>99.6</td>
</tr>
<tr>
<td>AE (newly trained)</td>
<td>9.2</td>
<td>10.0</td>
</tr>
<tr>
<td>Adv Training [4]</td>
<td>7.2</td>
<td>7.6</td>
</tr>
<tr>
<td>SOR [26]</td>
<td>18.8</td>
<td>17.2</td>
</tr>
<tr>
<td>DUP Net [26]</td>
<td>28</td>
<td>28.8</td>
</tr>
<tr>
<td>SRS [26]</td>
<td>43.2</td>
<td>29.2</td>
</tr>
</tbody>
</table>

Table A.2: Attacking Point Cloud Defenses (PointNet++ SSG): We evaluate untargeted attacks using norm-budgets of $\epsilon_{\infty} = 0.18$ and $\epsilon_{\infty} = 0.45$ with PointNet++ SSG [2] as the victim network under different defenses for 3D point clouds. Similar to before, we report attack success rates (higher indicates better attack). Note that both the attacks and evaluations are performed on PointNet++ SSG, which has an accuracy of 91.5% without input perturbations (for reference).

<table>
<thead>
<tr>
<th>Defenses</th>
<th>$\epsilon_{\infty} = 0.18$</th>
<th>$\epsilon_{\infty} = 0.45$</th>
</tr>
</thead>
<tbody>
<tr>
<td>No defense</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>AE (newly trained)</td>
<td>14.8</td>
<td>13.6</td>
</tr>
<tr>
<td>Adv Training [4]</td>
<td>12.0</td>
<td>7.6</td>
</tr>
<tr>
<td>SOR [26]</td>
<td>20.4</td>
<td>18.4</td>
</tr>
<tr>
<td>DUP Net [26]</td>
<td>18.0</td>
<td>16.4</td>
</tr>
<tr>
<td>SRS [26]</td>
<td>53.2</td>
<td>40.8</td>
</tr>
</tbody>
</table>
Table A.3: **Attacking Point Cloud Defenses (PointNet++ MSG):** We evaluate untargeted attacks using norm-budgets of $\epsilon_\infty = 0.18$ and $\epsilon_\infty = 0.45$ with PointNet++ MSG [2] as the victim network under different defenses for 3D point clouds. Similar to before, we report attack success rates (higher indicates better attack). Note that both the attacks and evaluations are performed on PointNet++ MSG, which has an accuracy of 91.5% without input perturbations (for reference).

<table>
<thead>
<tr>
<th>Defenses</th>
<th>$\epsilon_\infty = 0.18$</th>
<th>$\epsilon_\infty = 0.45$</th>
</tr>
</thead>
<tbody>
<tr>
<td>No defense</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>AE (newly trained)</td>
<td>13.2</td>
<td>10.0</td>
</tr>
<tr>
<td>Adv Training [4]</td>
<td>6.8</td>
<td>26.0</td>
</tr>
<tr>
<td>SOR [26]</td>
<td>21.6</td>
<td>26.0</td>
</tr>
<tr>
<td>DUP Net [26]</td>
<td>29.6</td>
<td>27.6</td>
</tr>
<tr>
<td>SRS [26]</td>
<td>43.6</td>
<td>45.6</td>
</tr>
</tbody>
</table>

Table A.4: **Attacking Point Cloud Defenses (PointNet):** We evaluate untargeted attacks using norm-budgets of $\epsilon_\infty = 0.18$ and $\epsilon_\infty = 0.45$ with PointNet [1] as the victim network under different defenses for 3D point clouds. Similar to before, we report attack success rates (higher indicates better attack). Note that both the attacks and evaluations are performed on PointNet, which has an accuracy of 92.8% without input perturbations (for reference).

<table>
<thead>
<tr>
<th>Defenses</th>
<th>$\epsilon_\infty = 0.18$</th>
<th>$\epsilon_\infty = 0.45$</th>
</tr>
</thead>
<tbody>
<tr>
<td>No defense</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>AE (newly trained)</td>
<td>8.0</td>
<td>7.6</td>
</tr>
<tr>
<td>Adv Training [4]</td>
<td>8.0</td>
<td>8.4</td>
</tr>
<tr>
<td>SOR [26]</td>
<td>16.0</td>
<td>15.6</td>
</tr>
<tr>
<td>DUP Net [26]</td>
<td>10.0</td>
<td>10.4</td>
</tr>
<tr>
<td>SRS [26]</td>
<td>80.8</td>
<td>81.6</td>
</tr>
</tbody>
</table>
Figure A.2: Ablation Study in $\ell_\infty$: Studying the effect of changing AdvPC hyperparameter ($\gamma$) on the success rate of the attack (left) and on its transferability (right). The transferability score reported for each victim network is the average success rate on the transfer networks averaged across all different norm-budgets $\epsilon_\infty$.

(averaged over all $\epsilon_\infty$ norm-budgets) in Figure A.2 for the four victim networks. One observation is that, while adding the AE loss with $\gamma > 0$ indeed improves transferability, it tends to deteriorate the success rate. We pick $\gamma = 0.25$ in our experiments to balance success and transferability.

A.4.2 Network Sensitivity to Point Cloud Attacks

Figure A.3 plot the sensitivity of the various networks when they are subject to input perturbations of varying norm-budgets $\epsilon_\infty$. We measure the classification accuracy of each network under our AdvPC attack ($\gamma = 0.25$), 3D-Adv [4], and KNN attack [5]. We observe that DGCNN [3] tends to be the most robust to adversarial perturbations in general. This might be explained by the fact that the convolution neighborhoods in DGCNN are dynamically updated across layers and iterations. This dynamic behavior in network structure may hinder the effect of the attack because gradient directions can change significantly from one iteration to another. This leads to failing attacks and higher robustness for DGCNN [3].
Figure A.3: **Sensitivity of Architectures in $\ell_\infty$:** We evaluate the sensitivity of each of the four networks for increasing norm-budget. For each network, we plot the classification accuracy under three attacks. Overall, DGCNN [3] is affected the least by adversarial perturbation.

### A.4.3 Effect of the Auto-Encoder (AE)

In Figure A.4, we show an example of how AE reconstruction preserves the details of the unperturbed point cloud and does not change the classifier prediction. When a perturbed point cloud passes through the AE, it recovers a natural-looking shape. The AE’s ability to reconstruct natural-looking 3D point clouds from various perturbed inputs might explain why it is a strong defense against attacks in Section A.3. Another observation from Figure A.4 is that when we fix the target $t'$ and do not enforce a specific incorrect target $t''$ (i.e. untargeted attack setting) for the data adversarial loss on the reconstructed point cloud in the AdvPC attack (Eq (2.6)), the optimization mechanism tends to pick $t''$ to be a similar class to the correct one. For example, a Toilet point cloud perturbed by AdvPC can be transformed into a Chair (similar in appearance to a toilet), if reconstructed by the AE. This effect is not observed for the other attacks [4, 5], which do not consider the data distribution and optimize solely for the network.

### A.4.4 Ablation Study on the Losses

We ablate each component of our pipeline and show their effect in our attacks. We evaluate this components by looking Attack Success Rate (ASR), transferability,
and the final norm obtained under the attack. In these experiments, we allow unconstrained attacks as well as constrained attacks. We show the effect of optimizing using EMD, CD, \( \ell_2 \), and \( \ell_\infty \). We show the results on Table A.5 for DGCNN network. We observe transferability is better when using hard constraints. Constraining the attack norm allows the optimization to learn more from the AE data distribution. The EMD doesn’t work well while the Chamfer loss is comparable to the \( \ell_2 \) loss.
### Attack Setup

<table>
<thead>
<tr>
<th>soft CD</th>
<th>soft EMD</th>
<th>soft $\ell_2$</th>
<th>hard $\ell_{\infty}$</th>
<th>hard $\ell_2$</th>
<th>AE</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Results

<table>
<thead>
<tr>
<th>CD</th>
<th>EMD</th>
<th>$\ell_{\infty}$</th>
<th>$\ell_2$</th>
<th>ASR</th>
<th>TR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.06</td>
<td>32.22</td>
<td>0.20</td>
<td>1.55</td>
<td>67.91</td>
<td>10.46</td>
</tr>
<tr>
<td>0.71</td>
<td>25.03</td>
<td>0.17</td>
<td>1.18</td>
<td>41.07</td>
<td>9.21</td>
</tr>
<tr>
<td>0.03</td>
<td>2.47</td>
<td>0.07</td>
<td>0.14</td>
<td>2.07</td>
<td>7.18</td>
</tr>
<tr>
<td>0.01</td>
<td>1.62</td>
<td>0.01</td>
<td>0.05</td>
<td>0.76</td>
<td>7.21</td>
</tr>
<tr>
<td>0.71</td>
<td>25.03</td>
<td>0.17</td>
<td>1.18</td>
<td>41.07</td>
<td>9.21</td>
</tr>
<tr>
<td>0.03</td>
<td>2.47</td>
<td>0.07</td>
<td>0.14</td>
<td>2.07</td>
<td>7.18</td>
</tr>
<tr>
<td>0.01</td>
<td>1.62</td>
<td>0.01</td>
<td>0.05</td>
<td>0.76</td>
<td>7.21</td>
</tr>
<tr>
<td>0.71</td>
<td>25.03</td>
<td>0.17</td>
<td>1.18</td>
<td>41.07</td>
<td>9.21</td>
</tr>
<tr>
<td>0.03</td>
<td>2.47</td>
<td>0.07</td>
<td>0.14</td>
<td>2.07</td>
<td>7.18</td>
</tr>
<tr>
<td>0.01</td>
<td>1.62</td>
<td>0.01</td>
<td>0.05</td>
<td>0.76</td>
<td>7.21</td>
</tr>
</tbody>
</table>

Table A.5: **Soft vs Hard on DGCNN**: study the effect of every bit of the loss on the norms, Attack Success Rate (ASR) and Transferability (TR) under unconstrained setup vs constrained setup in DGCNN [3]. ($\epsilon_{\infty} = 0.18$, $\epsilon_2 = 2.8$), $\lambda = 1$, $\gamma = 0.5$. **Bold** numbers are the best.
B Appendix B

B.1 Detailed Experimental Setup

B.1.1 Datasets

**ModelNet40.** We show in Figure B.1 examples of the mesh renderings of ModelNet40 used in training our MVTN. Note that the color of the object and the light direction are randomized in training for augmentation but are fixed in testing for stable performance.

**ShapeNet Core55.** In Figure B.2, we show examples of the point cloud renderings of ShapeNet Core55 [16, 20] used in training MVTN. Note how point cloud renderings offer more information about content hidden from the camera view-point, which can be useful for recognition. White color is used in training and testing for all point cloud renderings. For visualization purposes, colors are inverted in the main paper examples (Fig. 4 in the main paper).

**ScanObjectNN.** ScanObjectNN [17] has three main variants: object only, object with background, and the PB_T50_RS variant (hardest perturbed variant). Figure B.3 show examples of multi-view renderings of different samples of the dataset from its three variants. Note that adding the background points to the rendering gives some clues to our MVTN about the object, which explains why adding background improves the performance of MVTN in Table B.2.
B.1.2 MVTN Details

**MVTN Rendering.** Point cloud rendering offers a light alternative to mesh rendering in ShapeNet because its meshes contain large numbers of faces that hinder training the MVTN pipeline. Simplifying these 'high-poly' meshes (similar to ModelNet40) results in corrupted shapes that lose their main visual clues. Therefore, we use point cloud rendering for ShapeNet, allowing to process all shapes with equal memory requirements. Another benefit of point cloud rendering is making it possible to train MVTN with a large batch size on the same GPU (batch size of 30 on V100 GPU).

**MVTN Architecture.** We incorporate our MVTN into MVCNN [6] and ViewGCN [7]. In our experiments, we select PointNet [1] as the default point encoder of MVTN. All MVTNs and their baseline multi-view networks use ResNet18 [11] as the backbone in our main experiments with output feature size $d = 1024$. The azimuth angle maximum range ($u_{\text{bound}}$) is $\frac{180\degree}{M}$ for MVTN-circular and MVTN-spherical, while it is $180\degree$ for MVTN-direct. On the other hand, the elevation angle maximum range ($u_{\text{bound}}$) is $90\degree$. We use a 4-layer MLP for MVTN’s regression network $G$. For MVTN-spherical/MVTN-spherical, the regression network takes as input $M$ azimuth angles, $M$ elevation angles, and the point features of shape $S$ of size $b = 40$. The widths of the MVTN networks are illustrated in Figure B.4. MVTN concatenates all of its inputs, and the MLP outputs the offsets to the initial $2 \times M$ azimuth and elevation angles. The size of the MVTN network (with $b = 40$) is $14M^2 + 211M + 3320$ parameters, where $M$ is the number of views. It is a shallow network of only around 9K parameters when $M = 12$.

**View-Points.** In Figure B.5, we show the basic views configurations for $M$ views previously used in the literature: circular, spherical, and random. MVTN’s learned views are shown later in B.3.1 Since ViewGCN uses view sampling as a core operation, it requires the number of views to be at least 12, and hence, our MVTN with ViewGCN follows accordingly.
Training MVTN. We use AdamW [148] for our MVTN networks with a learning rate of 0.001. For other training details (e.g. training epochs and optimization), we follow the previous works [7, 6] for a fair comparison. The training of MVTN with MVCNN is done in 100 epochs and a batch size of 20, while the MVTN with ViewGCN is performed in two stages as proposed in the official code of the paper [7]. The first stage is 50 epochs of training the backbone CNN on the single view images, while the second stage is 35 epochs on the multi-view network on the $M$ views of the 3D object. We use learning rates of 0.0003 for MVCNN and 0.001 for ViewGCN, and a ResNet-18 [11] as the backbone CNN for both baselines and our MVTN-based networks. A weight decay of 0.01 is applied for both the multi-view network and the MVTN networks. Due to gradient instability from the renderer, we introduce gradient clipping in the MVTN to limit the $\ell_2$ norm of gradient updates to 30 for $G$. The code is available at https://github.com/ajhamdi/MVTN.
Figure B.1: **Training Data with Randomized Color and Lighting.** We show examples of mesh renderings of ModelNet40 used in training our MVTN. The color of the object and the light’s direction are randomized during training for augmentation purposes and fixed in testing for stable performance. For this figure, eight circular views are shown for each 3D shape.
Figure B.2: **ShapeNet Core55.** We show some examples of point cloud renderings of ShapeNet Core55 [16] used in training MVTN. Note how point cloud renderings offer more information about content hidden from the camera view-point (e.g. car wheels from the occluded side), which can be useful for recognition. For this figure, 12 spherical views are shown for each 3D shape.
Figure B.3: ScanObjectNN Variants. We show examples of point cloud renderings of different variants of the ScanObjectNN [17] point cloud dataset used to train MVTN. The variants are: object only, object with background, and the hardest perturbed variant (with rotation and translation). For this figure, six circular views are shown for each 3D shape.
Figure B.4: **MVTN Network Architecture.** We show a schematic and a code snippet for MVTN-spherical/MVTN-circular regression architectures used, where $b$ is the size of the point features extracted by the point encoder of MVTN and $M$ is the number of views learned. In most of our experiments, $b = 40$, while the output is the azimuth and elevation angles for all the $M$ views used. The network is drawn using [18]

```python
MVTN_regressor = Sequential(
    MLP([b+2*M, b, b, 3*M], activation="relu", dropout=0.5, batch_norm=True),
    MLP([2*M, 2*M], activation=None, dropout=0, batch_norm=False),
    nn.Tanh()
)
```

Figure B.5: **Views Configurations.** We show some possible view configurations that can be used with a varying number of views. (a): circular, (b): spherical, (c): random
B.2 Additional Results

B.2.1 Classification and Retrieval Benchmarks

We provide in Tables B.1, B.2, and B.3 comprehensive benchmarks of 3D classifications and 3D shape retrieval methods on ModelNet40 [19], ScanObjectNN [17], and ShapeNet Core55 [16, 20]. These tables include methods that use points as representations as well as other modalities like multi-view and volumetric representations. Our reported results of four runs are presented in each table as “max (avg ± std)”. Note in Table B.1 how our MVTN improves the previous state-of-the-art in classification (ViewGCN [7]) when tested on the same setup. Our implementations (highlighted using *) slightly differ from the reported results in their original paper. This can be attributed to the specific differentiable renderer of Pytorch3D [87] that we are using, which might not have the same quality of the non-differentiable OpenGL renderings [99] used in their setups.

B.2.2 Rotation Robustness

A common practice in the literature in 3D shape classification is to test the robustness of models trained on the aligned dataset by injecting perturbations during test time [30]. We follow the same setup as [30] by introducing random rotations during test time around the Y-axis (gravity-axis). We also investigate the effect of varying rotation perturbations on the accuracy of circular MVCNN when $M = 6$ and $M = 12$. We note from Figure B.6 that using less views leads to higher sensitivity to rotations in general. Furthermore, we note that our MVTN helps in stabilizing the performance on increasing thresholds of rotation perturbations.
Robustness to Y-Rotation in 3D Classification

Figure B.6: **Robustness on a Varying Y-Rotation.** We study the effect of varying the maximum rotation perturbation on the classification accuracies on ModelNet40. We compare the performance of circular MVCNN [6] to our circular-MVTN when it equips MVCNN when the number of views is 6 and 12. Note how MVTN stabilizes the drop in performance for larger Y-rotation perturbations, and the improvement is more significant for the smaller number of views \( M \).

### B.2.3 Occlusion Robustness

To quantify the occlusion effect due to the viewing angle of the 3D sensor in our setup of 3D classification, we simulate realistic occlusion by cropping the object from canonical directions. We train PointNet [1], DGCNN [3], and MVTN on the ModelNet40 point cloud dataset. Then, at test time, we crop a portion of the object (from 0% occlusion ratio to 75%) along the \( \pm X \), \( \pm Y \), and \( \pm Z \) directions independently. Figure B.8 shows examples of this occlusion effect with different occlusion ratios. We report the average test accuracy (on all the test set) of the six cropping directions for the baselines and MVTN in Figure B.7. Note how MVTN achieves high test accuracy even when large portions of the object are cropped. Interestingly, MVTN outperforms PointNet [1] by 13% in test accuracy when half of the object is occluded.
Figure B.7: **Occlusion Robustness of 3D Methods.** We plot test accuracy vs. the Occlusion Ratio of the data to simulate the occlusion robustness of different 3D methods: PointNet [1], DGCNN [3], and MVTN. Our MVTN achieves close to 13% better than PointNet when half of the object is occluded. MVTN\(^1\) refers to MVTN with MVCNN as the multi-view network while MVTN\(^2\) refers to MVTN with View-GCN as the multi-view network.

This result is significant, given that PointNet is well-known for its robustness [1, 101].
Figure B.8: **Occlusion of 3D Objects**: We simulate realistic occlusion scenarios in 3D point clouds by cropping a percentage of the object along canonical directions. Here, we show an object occluded with different ratios and from different directions.
<table>
<thead>
<tr>
<th>Method</th>
<th>Data Type</th>
<th>Classification Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(Per-Class)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Overall)</td>
</tr>
<tr>
<td>SPH [149]</td>
<td>Voxels</td>
<td>68.2</td>
</tr>
<tr>
<td>LFD [95]</td>
<td>Voxels</td>
<td>75.5</td>
</tr>
<tr>
<td>3D ShapeNets [19]</td>
<td>Voxels</td>
<td>77.3</td>
</tr>
<tr>
<td>VoxNet [58]</td>
<td>Voxels</td>
<td>83.0</td>
</tr>
<tr>
<td>VRN [150]</td>
<td>Voxels</td>
<td>-</td>
</tr>
<tr>
<td>MVCNN-MS [151]</td>
<td>Voxels</td>
<td>-</td>
</tr>
<tr>
<td>FusionNet [152]</td>
<td>Voxels+MV</td>
<td>-</td>
</tr>
<tr>
<td>PointNet [1]</td>
<td>Points</td>
<td>86.2</td>
</tr>
<tr>
<td>PointNet++ [2]</td>
<td>Points</td>
<td>-</td>
</tr>
<tr>
<td>KD-Network [153]</td>
<td>Points</td>
<td>88.5</td>
</tr>
<tr>
<td>PointCNN [44]</td>
<td>Points</td>
<td>88.1</td>
</tr>
<tr>
<td>DGCNN [3]</td>
<td>Points</td>
<td>90.2</td>
</tr>
<tr>
<td>KPConv [86]</td>
<td>Points</td>
<td>-</td>
</tr>
<tr>
<td>PVNet [85]</td>
<td>Points</td>
<td>-</td>
</tr>
<tr>
<td>PTransformer [90]</td>
<td>Points</td>
<td>90.6</td>
</tr>
<tr>
<td>MVCNN [6]</td>
<td>12 Views</td>
<td>90.1</td>
</tr>
<tr>
<td>GVCNN [67]</td>
<td>12 Views</td>
<td>90.7</td>
</tr>
<tr>
<td>ViewGCN [7]</td>
<td>20 Views</td>
<td>96.5</td>
</tr>
<tr>
<td>ViewGCN [7]*</td>
<td>12 views</td>
<td>90.7 (90.5 ± 0.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>93.0 (92.8 ± 0.1)</td>
</tr>
<tr>
<td>ViewGCN [7]*</td>
<td>20 views</td>
<td>91.3 (91.0 ± 0.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>93.3 (93.1 ± 0.2)</td>
</tr>
<tr>
<td>MVTN (ours)*</td>
<td>12 Views</td>
<td>92.0 (91.2 ± 0.6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>93.8 (93.4 ± 0.3)</td>
</tr>
<tr>
<td>MVTN (ours)*</td>
<td>20 Views</td>
<td>92.2 (91.8 ± 0.3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>93.5 (93.1 ± 0.5)</td>
</tr>
</tbody>
</table>

Table B.1: **3D Shape Classification on ModelNet40.** We compare MVTN against other methods in 3D classification on ModelNet40 [19]. * indicates results from our rendering setup (differentiable pipeline), while other multi-view results are reported from pre-rendered views. **Bold** denotes the best result in its setup. In brackets, we report the average and standard deviation of four runs.
<table>
<thead>
<tr>
<th>Method</th>
<th>Object with Background</th>
<th>Object Only</th>
<th>PB_T50_RS (Hardest)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DMFV [91]</td>
<td>68.2</td>
<td>73.8</td>
<td>63.0</td>
</tr>
<tr>
<td>PointNet [1]</td>
<td>73.3</td>
<td>79.2</td>
<td>68.0</td>
</tr>
<tr>
<td>SpiderCNN [92]</td>
<td>77.1</td>
<td>79.5</td>
<td>73.7</td>
</tr>
<tr>
<td>PointNet ++ [2]</td>
<td>82.3</td>
<td>84.3</td>
<td>77.9</td>
</tr>
<tr>
<td>PointCNN [44]</td>
<td>86.1</td>
<td>85.5</td>
<td>78.5</td>
</tr>
<tr>
<td>DGCNN [3]</td>
<td>82.8</td>
<td>86.2</td>
<td>78.1</td>
</tr>
<tr>
<td>SimpleView [93]</td>
<td>-</td>
<td>-</td>
<td>79.5</td>
</tr>
<tr>
<td>BGA-DGCNN [17]</td>
<td>-</td>
<td>-</td>
<td>79.7</td>
</tr>
<tr>
<td>BGA-PN++ [17]</td>
<td>-</td>
<td>-</td>
<td>80.2</td>
</tr>
<tr>
<td>ViewGCN *</td>
<td>91.9 (91.12 ± 0.5)</td>
<td>90.4 (89.7 ± 0.5)</td>
<td>80.5 (80.2 ± 0.4)</td>
</tr>
<tr>
<td>MVTN (ours)</td>
<td><strong>92.6</strong> (92.5 ± 0.2)</td>
<td><strong>92.3</strong> (91.7 ± 0.7)</td>
<td><strong>82.8</strong> (81.8 ± 0.7)</td>
</tr>
</tbody>
</table>

Table B.2: 3D Point Cloud Classification on ScanObjectNN. We compare the performance of MVTN in 3D point cloud classification on three different variants of ScanObjectNN [17]. The variants include object with background, object only, and the hardest variant. * indicates results from our rendering setup (differentiable pipeline), and we report the average and standard deviation of four runs in brackets.

<table>
<thead>
<tr>
<th>Method</th>
<th>Data Type</th>
<th>ModelNet40</th>
<th>ShapeNet Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZDFR [154]</td>
<td>Voxels</td>
<td>-</td>
<td>19.9</td>
</tr>
<tr>
<td>DLAN [84]</td>
<td>Voxels</td>
<td>-</td>
<td>66.3</td>
</tr>
<tr>
<td>SPH [149]</td>
<td>Voxels</td>
<td>33.3</td>
<td>-</td>
</tr>
<tr>
<td>LFD [95]</td>
<td>Voxels</td>
<td>40.9</td>
<td>-</td>
</tr>
<tr>
<td>3D ShapeNets [19]</td>
<td>Voxels</td>
<td>49.2</td>
<td>-</td>
</tr>
<tr>
<td>PVNet [85]</td>
<td>Points</td>
<td>89.5</td>
<td>-</td>
</tr>
<tr>
<td>MVCNN [6]</td>
<td>12 Views</td>
<td>80.2</td>
<td>73.5</td>
</tr>
<tr>
<td>GIFT [97]</td>
<td>20 Views</td>
<td>-</td>
<td>64.0</td>
</tr>
<tr>
<td>MVFusionNet [98]</td>
<td>12 Views</td>
<td>-</td>
<td>62.2</td>
</tr>
<tr>
<td>ReVGG [20]</td>
<td>20 Views</td>
<td>-</td>
<td>74.9</td>
</tr>
<tr>
<td>RotNet [8]</td>
<td>20 Views</td>
<td>-</td>
<td>77.2</td>
</tr>
<tr>
<td>ViewGCN [7]</td>
<td>20 Views</td>
<td>-</td>
<td>78.4</td>
</tr>
<tr>
<td>MLVCNN [80]</td>
<td>24 Views</td>
<td>92.2</td>
<td>-</td>
</tr>
<tr>
<td>MVTN (ours)</td>
<td>12 Views</td>
<td><strong>92.9</strong> (92.4 ± 0.6)</td>
<td><strong>82.9</strong> (82.4 ± 0.6)</td>
</tr>
</tbody>
</table>

Table B.3: 3D Shape Retrieval. We benchmark the shape retrieval capability of MVTN on ModelNet40 [19] and ShapeNet Core55 [16, 20]. MVTN achieves the best retrieval performance among recent state-of-the-art methods on both datasets with only 12 views. In brackets, we report the average and standard deviation of four runs.
B.3 Analysis and Insights

B.3.1 Ablation Study

This section introduces a comprehensive ablation study on the different components of MVTN, and their effect on test accuracy on the standard ModelNet40 [19].

**MVTN Variants.** We study the effect of the number of views $M$ on the performance of different MVTN variants (direct, circular, spherical). The experiments are repeated four times, and the average test accuracies with confidence intervals are shown in Figure B.9. The plots show how learned MVTN-spherical achieves consistently superior performance across a different number of views. Also, note that MVTN-direct suffers from over-fitting when the number of views is larger than four (i.e. it gets perfect training accuracy but deteriorates in test accuracy). This can be explained by observing that the predicted view-points tend to be similar to each other for MVTN-direct when the number of views is large. The similarity in views leads the multi-view network to memorize the training but to suffer in testing.

**Backbone.** In the main manuscript (Table 6), we study MVTN with ViewGCN as the multi-view network. Here, we study the backbone effect on MVTN with MVCNN as the multi-view network and report all results in Table B.5. The study includes the backbone choice, and the point encoder choice. Note that including more sophisticated backbones does not improve the accuracy

**Late Fusion.** In the MVTN pipeline, we use a point encoder and a multi-view network. One can argue that an easy way to combine them would be to fuse them later in the architecture. For example, PointNet [1] and MVCNN [6] can be max pooled together at the last layers and trained jointly. We train such a setup and compare it to MVTN. We observe that MVTN achieves 91.8% compared to 88.4% by late fusion. More results are reported in Table B.5

**Effect of Object Color.** Our main experiments used random colors for the objects
during training and fixed them to white in testing. We tried different coloring approaches, like using a fixed color during training and test. The results are illustrated in Table B.4.

**Image size and number of points.** We study the effect of rendered image size and the number of points sampled in a 4-view MVTN trained on ModelNet40 and report the overall accuracies (averaged over four runs) as follows. For image sizes 160×160, 224×224, and 280×280, the results are 91.0%, 91.6%, and 91.9% respectively. For the number of randomly sampled points \( P = 512, 1024, \) and 2048, the results are 91.2% 91.6%, and 91.6% respectively.

**Learning Distance to the Object.** One possible ablation to the MVTN is to learn the distance to the object. This feature should allow the cameras to get closer to details that might be important to the classifier to understand the object properly. However, we observe that MVTN generally performs worse or does not improve with this setup, and hence, we refrain from learning it. In all of our main experiments, we fixed the distance to 2.2 units, which is a good middle ground providing best accuracy. Please see Figure B.10 for the effect of picking a fixed distance in training spherical ViewGCN.

<table>
<thead>
<tr>
<th>Method</th>
<th>Object Color</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>White</td>
<td>Random</td>
<td></td>
</tr>
<tr>
<td>Fixed views</td>
<td>92.8 ± 0.1</td>
<td>92.8 ± 0.1</td>
<td></td>
</tr>
<tr>
<td>MVTN (learned)</td>
<td>93.3 ± 0.1</td>
<td><strong>93.4 ± 0.1</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table B.4: **Effect of Color Selection.** We ablate selecting the color of the object in training our MVTN and when views are fixed in the spherical configuration. Fixed white color is compared to random colors in training. Note how randomizing the color helps in improving the test accuracy on ModelNet40 a little bit.
Figure B.9: **Variants of MVTN.** We plot test accuracy vs. the number of views used in training different variants of our MVTN. Note how MVTN-spherical is generally more stable in achieving better performance on ModelNet40. 95% confidence interval is also plotted on each setup (repeated four times).

**B.3.2 Transferability of MVTN View-Points**

We hypothesize that the views learned by MVTN are transferable across multi-view classifiers. We believe MVTN picks the best views based on the actual shape and is less influenced by the multi-view network. This means that MVTN learns views that are more representative of the object, making it easier for any multi-view network to recognize it. As such, we ask the following: *can we transfer the views MVTN learns under one setting to a different multi-view network?*

To test our hypothesis, we take a 12-view MVTN-spherical module trained with MVCNN as a multi-view network and transfer the predicted views to a ViewGCN multi-view network. In this case, we freeze the MVTN module and only train ViewGCN on these learned but fixed views. ViewGCN with transferred MVTN views reaches 93.1% accuracy in classification. It corresponds to a boost of 0.7% from the 92.4% of the original ViewGCN. Although this result is lower than fully
Figure B.10: **Effect of Distance to 3D Object.** We study the effect of changing the distance on training a spherical ViewGCN. We show that the distance of 2.2 units to the center is in between far and close it and gives the best accuracy.

trained MVTN($-0.3\%$), we observe a decent transferability between both multi-view architectures.

### B.3.3 Shape Retrieval Examples

We show qualitative examples of our retrieval results using the MVTN-spherical with ViewGCN in Figure B.11. Note that the top ten retrieved objects for all these queries are positive (from the same classes of the queries).
Table B.5: **Ablation Study.** We study the effect of ablating different components of MVTN on the test accuracy on ModelNet40. Namely, we observe that using more complex backbone CNNs (like ResNet50 [11]) or a more complex features extractor (like DGCNN [3]) does not increase the performance significantly compared to ResNet18 and PointNet [1] respectively. Furthermore, combining the shape features extractor with the MVCNN [6] in late fusion does not work as well as MVTN with the same architectures. All the reported results are using MVCNN [6] as multi-view network.
Figure B.11: **Qualitative Examples for Object Retrieval:** *(left):* we show some query objects from the test set. *(right):* we show top ten retrieved objects by our MVTN from the training set.
C Appendix C

C.1 Analyzing Deep Neural Networks

Here we visualize different Network semantic maps generated during our analysis. In the 1D case, we fix the elevation of the camera to a nominal angle of 35° and rotate around the object. In the 2D case, we change both the elevation and azimuth around the object. These maps can be generated to any type of semantic parameters that affect the generation of the image, and not viewing angle.

C.1.1 Networks Semantic Maps (1D)

In Figure C.1 we visualize the 1D semantic maps of rotating around the object and recording different DNNs performance and averaging the profile over 10 different shapes per class.

C.1.2 Networks Semantic Maps (2D)

In Figure C.2 we visualize the 2D semantic maps of elevation angles and rotating around the object and recording different DNNs performances and averaging the maps over 10 different shapes per class.

C.1.3 Examples of Found Regions ( with Example Renderings)

In Figure C.3, we provide examples of 2D regions found with the three algorithms along with renderings of the shapes from the robust regions detected.
C.1.4 Analyzing Semantic Data Bias in ImageNet

In Figure C.4, we visualize semantic data bias in the common training dataset (i.e. ImageNet [12]) by averaging the Networks Semantic Maps (NSM) of different networks and on different shapes, Different classes have a different semantic bias in ImageNet as clearly shown in the maps above. These places of high confidence probably reveal the areas where an abundance of examples exists in ImageNet, while holes convey scarcity of such examples in ImageNet’s corresponding class.

C.2 Detailed Derivations of the Update Directions of the Bounds

C.2.1 Defining Robustness as an Operator

In our case we consider a more general case where we are interested in the $u \in \Omega \subset \mathbb{R}^n$, a hidden latent parameter that generates the image and passes to scene generator (e.g. a renderer function $R$) that takes the parameter $u$ and an object shape $S$ of a class that is identified by classifier $C$. $\Omega$ is the continuous semantic space for the parameters that we intend to study. The renderer creates the image $x \in \mathbb{R}^d$, and then we study the behavior of a classifier $C$ of that image across multiple shapes and multiple famous DNNs. Now, this function of interest is defined as follows.

$$f(u) = C_z(R(S_z, u)) , \quad 0 \leq f(u) \leq 1 \quad (C.1)$$

Where $z$ is a class label of interest of study, and we observe the network score for that class by rendering a shape $S_z$ of the same class. The shape and class labels are constants, and only the parameters vary for $f$. The robust-region-finding operator is then defined as follows.

$$\Phi_{\text{robust}}(f(u), S_z, u_0) = \mathbb{D} = \{ u : a \leq u \leq b \}$$

s.t. $\mathbb{E}_{u \sim \mathbb{D}}[f(u)] \geq 1 - \epsilon_m$, $u_0 \in \mathbb{D}$, $\text{VAR}[f(u)] \leq \epsilon_v \quad (C.2)$
where the left and right bounds of $\mathbb{D}$ are $a = [a_1, a_2, ..., a_n]$ and $b = [b_1, b_2, ..., b_n]$ respectively. The two small thresholds $\epsilon_m, \epsilon_v$ are to insure high performance and low variance of the DNN network in that robust region. We can define the opposite operator which is to find adversarial regions like follows :

$$\Phi_{\text{adv}}(f(u), S_z, u_0) = \mathbb{D} = \{u : a \leq u \leq b\}$$

s.t. $E_{u \sim \mathbb{D}}[f(u)] \leq \epsilon_m$, $u_0 \in \mathbb{D}$, $\text{VAR}[f(u)] \geq \epsilon_v$ \hspace{1cm} (C.3)

We can show clearly that $\Phi_{\text{adv}}$ and $\Phi_{\text{robust}}$ are related as follows

$$\Phi_{\text{adv}}(f(u), S_z, u_0) = \Phi_{\text{robust}}(1 - f(u), S_z, u_0)$$ \hspace{1cm} (C.4)

So we can just focus our attentions on $\Phi_{\text{robust}}$ to find robust regions , and the adversarial regions follow directly from Eq (C.4).

### C.2.2 Divergence of the Bounds

To develop an algorithm for $\Phi$, we deploy the idea by [117] which focus on maximizing the inner area of the function in the region and fitting the bounds to grow the region’s bounds. As we will show, maximizing the region by maximizing the integral can lead to divergence, as follows :

**Lemma 1.** Let $f$ be a continuous scalar function $f : \mathbb{R}^1 \rightarrow \mathbb{R}^1$, and let $L$ be the function defining the definite integral of $f$ in terms of the two integral bounds , i.e. $L(a, b) = \int_a^b f(u)du$. Then, to maximize $L$, $-f(a)$ and $f(b)$ are valid ascent directions for the two bounds $a, b$ respectively.

**Proof.** a direction $p$ is an ascent direction of objective $L$ if it satisfies the inequality $p^T \nabla L \geq 0$.\[155].

To find $\frac{\partial L}{\partial a} = \frac{\partial}{\partial a} \int_a^b f(u)du$, we use the Leibniz rule from the fundamental theorem of calculus which states that $\frac{d}{dx} \int_a^{b(x)} f(u)du = f(b(x)) \frac{d}{dx} b(x) - f(a(x)) \frac{d}{dx} a(x)$
Therefore, $\frac{\partial}{\partial a} \int_a^b f(u)\,du = f(b) \times 0 - f(a) \times 1 = -f(a)$. Similarly, $\frac{\partial}{\partial b} f(u)\,du = f(b)$.

By picking $p = [-f(a), f(b)]^T$, then $p^T \nabla L = f(a)^2 + f(b)^2 \geq 0$. This proves that $p$ is a valid ascent direction for objective $L$.

**Theorem 1.** Let $f$ be a positive continuous scalar function $f : \mathbb{R}^1 \rightarrow (0, 1)$, and let $L$ be the function defining the definite integral of $f$ in terms of the two integral bounds, i.e. $L(a, b) = \int_a^b f(u)\,du$. Then, following the ascent direction in Lemma 1 can diverge the bounds if followed in a gradient ascent technique with fixed learning rate.

**Proof.** If we follow the direction $[-f(a), f(b)]^T$, with a fixed learning rate $\eta$, then the update rules for $a, b$ will be as follows. $a_k = a_{k-1} - \eta f(a), \quad b_k = b_{k-1} + \eta f(b)$. For initial points $a_0, b_0$, then $a_k = a_0 - \eta \sum_{i=0}^k f(Area_{in}), \quad b_k = b_0 + \eta \sum_{i=0}^k f(b_i)$. We can see now if $f(u) = c, 0 < c < 1$, then as $k \rightarrow \infty$, the bounds $a_k \rightarrow -\infty, \quad b_k \rightarrow \infty$. This leads to the claimed divergence.

To solve the issue of bounds diverging we propose the following formulations for one-dimensional bounds, and then we extend them to n- dimensions, which allows for finding the n-dimensional semantic robust/adversarial regions that are similar to figure 2. Some of the formulations are black-box in nature (they don’t need the gradient of the function $f$ in order to update the current estimates of the bound) while others.

**C.2.3 Naive Approach**

$$L = -\text{Area}_{in} + \frac{\lambda}{2} |b - a|^2$$  \hspace{1cm} (C.5)

Using Leibniz rule as in Lemma 1, we get the following update steps for the objective $L$:

$$\frac{\partial L}{\partial a} = f(a) - \lambda (b-a)$$

$$\frac{\partial L}{\partial b} = -f(b) + \lambda (b-a)$$  \hspace{1cm} (C.6)
where $\lambda$ is regularizing the boundaries not to extend too much in the case the function evaluation was positive all the time.

**Extension to $n$-dimension:** Let’s start with $n = 2$. Now, we have $f : \mathbb{R}^2 \to (0, 1)$, then we define the loss integral as a function of four bounds of a rectangular region as follows.

$$L(a_1, a_2, b_1, b_2) =$$

$$- \int_{a_2}^{b_2} \int_{a_1}^{b_1} f(u, v) dv du + \frac{\lambda}{2} |b_1 - a_1|^2 + \frac{\lambda}{2} |b_2 - a_2|^2 \tag{C.7}$$

We apply the trapezoidal approximation on the loss to obtain the following expression.

$$L(a_1, a_2, b_1, b_2) \approx \frac{\lambda}{2} |b_1 - a_1|^2 + \frac{\lambda}{2} |b_2 - a_2|^2$$

$$- \frac{(b_1 - a_1)(b_2 - a_2)}{4} \left( f(a_1, a_2) + f(b_1, a_2) + f(a_1, b_2) + f(b_1, b_2) \right) \tag{C.8}$$

to find the update direction for the first bound $a_1$ by taking the partial derivative of the function in Eq (C.7) we get the following update direction for $a_1$ along with its trapezoidal approximation in order to able to compute it during the optimization:

$$\frac{\partial L}{\partial a_1} = \int_{a_2}^{b_2} f(a_1, v) dv - \lambda(b_1 - a_1)$$

$$\approx \frac{(b_2 - a_2)}{2} \left( f(a_1, a_2) + f(a_1, b_2) \right) - \lambda(b_1 - a_1) \tag{C.9}$$

Doing similar steps to the first bound for the other three bounds we obtain the full update directions for $(a_1, a_2, b_1, b_2)$

$$\frac{\partial L}{\partial a_1} \approx \frac{(b_2 - a_2)}{2} \left( f(a_1, a_2) + f(a_1, b_2) \right) - \lambda(b_1 - a_1)$$

$$\frac{\partial L}{\partial b_1} \approx -\frac{(b_2 - a_2)}{2} \left( f(b_1, a_2) + f(b_1, b_2) \right) + \lambda(b_1 - a_1) \tag{C.10}$$

$$\frac{\partial L}{\partial a_2} \approx \frac{(b_1 - a_1)}{2} \left( f(a_1, a_2) + f(b_1, a_2) \right) - \lambda(b_2 - a_2)$$

$$\frac{\partial L}{\partial b_2} \approx -\frac{(b_1 - a_1)}{2} \left( f(a_1, b_2) + f(b_1, b_2) \right) + \lambda(b_2 - a_2)$$
Where $f_1'(.) = \frac{\partial f(u,v)}{\partial u}$, $f_2'(.) = \frac{\partial f(u,v)}{\partial v}$. Now, for $f : \mathbb{R}^n \rightarrow (0, 1)$, we define the inner region hyper-rectangle as before $\mathcal{D} = \{x : a \leq x \leq b\}$. Here, we assume the size of the region is positive at every dimension, i.e. $r = b - a > 0$. The volume of the region $\mathcal{D}$ normalized by exponent of dimension $n$ is expressed as follows

$$\text{volume}(\mathcal{D}) = \Delta = \frac{1}{2^n} \prod_{i=1}^{n} r_i \quad (C.11)$$

The region $\mathcal{D}$ can also be defined in terms of the matrix $\mathbf{D}$ of all the corner points $\{\mathbf{d}^i\}^{2^n}_{i=1}$ as follows.

$$\text{corners}(\mathcal{D}) = \mathbf{D}_{n \times 2^n} = \begin{bmatrix} \mathbf{d}^1 | \mathbf{d}^2 | \ldots | \mathbf{d}^{2^n} \end{bmatrix}$$

$$\mathbf{D} = \mathbf{1}^T \mathbf{a} + \mathbf{M}^T \odot (\mathbf{1}^T \mathbf{r}) \quad (C.12)$$

where $\mathbf{1}$ is the all-ones vector of size $2^n$, $\odot$ is the Hadamard product of matrices (element-wise), and $\mathbf{M}$ is a constant masking matrix defined as the matrix of binary numbers of $n$ bits that range from 0 to $2^n - 1$ defined as follows.

$$\mathbf{M}_{n \times 2^n} = \begin{bmatrix} \mathbf{m}^0 | \mathbf{m}^1 | \ldots | \mathbf{m}^{2^n-1} \end{bmatrix}, \text{ where } \mathbf{m}^i = \text{binary}_n(i) \quad (C.13)$$

We define the function vector as the vector $\mathbf{f}_\mathcal{D}$ of all function evaluations at all corner points of $\mathcal{D}$

$$\mathbf{f}_\mathcal{D} = \begin{bmatrix} f(\mathbf{d}^1), f(\mathbf{d}^2), \ldots, f(\mathbf{d}^{2^n}) \end{bmatrix}^T, \quad \mathbf{d}^i = \mathbf{D}(:,i) \quad (C.14)$$

We follow similar steps as in $n = 2$ and obtain the following loss expressions and
update directions:

\[ L(a, b) = -\int \cdots \int d_D f(u_1, \ldots, u_n) \, du_1 \ldots du_n + \frac{\lambda}{2} |r|^2 \]

\[ \approx -\Delta^T f_d + \frac{\lambda}{2} |r|^2 \]  \hspace{1cm} (C.15)

\[ \nabla_a L \approx 2\Delta \text{diag}^{-1}(r)M_d + \lambda r \]

\[ \nabla_b L \approx -2\Delta \text{diag}^{-1}(r)M_d - \lambda r \]

C.2.4 Outer-Inner Ratio Loss (OIR)

We introduce an outer region \( A, B \) with the bigger area that contains the small region \((a, b)\). We follow the following assumption to insure that the outer area is always positive.

\[ A = a - \alpha \frac{b-a}{2}, \quad B = b + \alpha \frac{b-a}{2} \]  \hspace{1cm} (C.16)

where \( \alpha \) is the small boundary factor of the outer area to the inner area. We formulate the problem as a ratio of outer over the inner area and we try to make this ratio as close as possible to 0 . \( L = \frac{\text{Area_{out}}}{\text{Area_{in}}} \) We use the DencklBeck technique for solving non-linear fractional programming problems [121] to transform \( L \) as follows.

\[ L = \frac{\text{Area}_{\text{out}}}{\text{Area}_{\text{in}}} = \text{Area}_{\text{out}} - \lambda \text{Area}_{\text{in}} \]

\[ = \int_B f(a) \, da - \int_a^b f(a) \, da - \lambda \int_a^b f(a) \, da \]  \hspace{1cm} (C.17)

where \( \lambda^* = \frac{\text{Area}_{\text{out}}}{\text{Area}_{\text{in}}} \) is the DencklBeck factor and it is equal to the small objective best achieved.

Black-Box (OIR_B).

Here we set \( \lambda = 1 \) to simplify the problem. This yields the following expression of
the loss
\[ L = \text{Area}_{\text{out}} - \text{Area}_{\text{in}} \]
\[ = \int_{a}^{b} f(u)du + \int_{A}^{B} f(u)du - \int_{a}^{b} f(u)du \]
\[ = \int_{A}^{B} f(u)du - 2 \int_{a}^{b} f(u)du \]
\[ = \int_{a - \frac{b - a}{2}}^{b + \alpha \frac{b - a}{2}} f(u)du - 2 \int_{a}^{b} f(u)du \]  
(C.18)

using Leibniz rule as in Lemma 1, we get the following update steps for the objective \( L \):
\[ \frac{\partial L}{\partial a} = - (1 + \alpha \frac{1}{2}) f(A) - \alpha \frac{1}{2} f(B) + 2 f(a) \]
\[ \frac{\partial L}{\partial b} = (1 + \alpha \frac{1}{2}) f(B) + \alpha \frac{1}{2} f(A) - 2 f(b) \]  
(C.19)

Extension to \( n \)-dimension: Let's start by \( n = 2 \). Now, we have \( f : \mathbb{R}^2 \rightarrow (0, 1) \), and with the following constrains on the outer region.
\[ A_1 = a_1 - \frac{b_1 - a_1}{2}, \quad B_1 = b_1 + \frac{b_1 - a_1}{2} \]
\[ A_2 = a_2 - \frac{b_2 - a_2}{2}, \quad B_1 = b_2 + \frac{b_2 - a_2}{2} \]  
(C.20)

we define the loss integral as a function of four bounds of a rectangular region as follows.
\[ L(a_1, a_2, b_1, b_2) = \]
\[ \int_{A_2}^{B_2} \int_{A_1}^{B_1} f(u, v)dvdu - 2 \int_{a_2}^{b_2} \int_{a_1}^{b_1} f(u, v)dvdu \]  
(C.21)
We apply the trapezoidal approximation on the loss to obtain the following expression.

\[
L(a_1, a_2, b_1, b_2) \approx \frac{(B_1 - A_1)(B_2 - A_2)}{4} ( f(A_1, A_2) + f(B_1, A_2) + f(A_1, B_2) + f(B_1, B_2) ) \\
- \frac{(b_1 - a_1)(b_2 - a_2)}{2} ( f(a_1, a_2) + f(b_1, a_2) + f(a_1, b_2) + f(b_1, b_2) ) \\
= \frac{(b_1 - a_1)(b_2 - a_2)}{4} (1 + \alpha)^2 ( f(A_1, A_2) + f(B_1, A_2) + f(A_1, B_2) + f(B_1, B_2) ) \\
- 2( f(a_1, a_2) + f(b_1, a_2) + f(a_1, b_2) + f(b_1, b_2) ) 
\]

(C.22)

Doing similar steps to the first bound for the other three bounds we obtain the full

\[
\frac{\partial L}{\partial a_1} = -(1 + \alpha) \int_{A_2}^{B_2} ( f(A_1, v) + \frac{\alpha}{2} f(B_1, v) ) dv \\
+ 2 \int_{a_2}^{b_2} f(a_1, v) dv \\
\approx \frac{(b_2 - a_2)}{2} (1 + \alpha)(1 + \frac{\alpha}{2})(f(A_1, A_2) + f(A_1, B_2)) \\
+ \frac{\alpha}{2} (f(B_1, A_2) + f(B_1, B_2)) \\
+ 2( f(a_1, a_2) + f(a_1, b_2) ) 
\]

(C.23)
update directions for \((a_1, a_2, b_1, b_2)\)

\[
\frac{\partial L}{\partial a_1} \approx \frac{b_2 - a_2}{2} (1 + \alpha)(1 + \frac{\alpha}{2})(f(A_1, A_2) + f(A_1, B_2)) + \frac{\alpha}{2} (f(B_1, A_2) + f(B_1, B_2)) \\
+ 2\left( f(a_1, a_2) + f(a_1, b_2) \right)
\]  
(C.24)

\[
\frac{\partial L}{\partial b_1} \approx \frac{b_2 - a_2}{2} (1 + \alpha)(1 + \frac{\alpha}{2})(f(B_1, A_2) + f(B_1, B_2)) + \frac{\alpha}{2} (f(A_1, A_2) + f(A_1, B_2)) \\
- 2\left( f(b_1, a_2) + f(b_1, b_2) \right)
\]  
(C.25)

\[
\frac{\partial L}{\partial a_2} \approx \frac{b_1 - a_1}{2} (1 + \alpha)(1 + \frac{\alpha}{2})(f(A_1, A_2) + f(B_1, B_2)) + \frac{\alpha}{2} (f(A_1, B_2) + f(B_1, B_2)) \\
+ 2\left( f(a_1, a_2) + f(b_1, a_2) \right)
\]  
(C.26)

\[
\frac{\partial L}{\partial b_2} \approx \frac{b_1 - a_1}{2} (1 + \alpha)(1 + \frac{\alpha}{2})(f(A_1, B_2) + f(B_1, B_2)) + \frac{\alpha}{2} (f(A_1, A_2) + f(B_1, A_2)) \\
- 2\left( f(a_1, b_2) + f(b_1, b_2) \right)
\]  
(C.27)

Where \(f_i'(.), f_i''(.) = \frac{\partial f(u,v)}{\partial u}, \frac{\partial f(u,v)}{\partial v}\) Now, for \(f : \mathbb{R}^n \to (0,1)\), we define the inner region hyper-rectangle as before \(D = \{x : a \leq x \leq b\}\), but now define an outer bigger region \(Q\) that include the smaller region \(D\) and defined as follows:

\[
Q = \{x : a - \frac{\alpha}{2}r \leq x \leq b + \frac{\alpha}{2}r\},\text{ where } a, b, r \text{ are defined as before, while } \alpha \text{ is}
\]
defined as the boundary factor of the outer region in all the dimensions equivalently. The inner and outer regions can also be defined in terms of the corner points as follows.

$$\text{corners}(D) = D_{n \times 2^n} = [d^1|d^2|...|d^{2^n}]$$
$$D = 1^T a + M^T \odot (1^T r)$$

$$\text{corners}(Q) = Q_{n \times 2^n} = [q^1|q^2|...|q^{2^n}]$$
$$Q = 1^T (a - \frac{\alpha}{2} r) + (1 + \alpha)M^T \odot (1^T r)$$

where 1 is the all-ones vector of size $2^n$, $\odot$ is the Hadamard product of matrices (element-wise), and $M$ is a constant masking matrix defined in Eq (C.13). Now we define two function vectors evaluated at all possible inner and outer corner points respectively.

$$f_D = [f(d^1), f(d^2), ..., f(d^{2^n})]^T, \quad d^i = D_{:,i}$$
$$f_Q = [f(q^1), f(q^2), ..., f(q^{2^n})]^T, \quad c^i = Q_{:,i}$$

(C.29)

Now the loss and update directions for the n-dimensional case becomes as follows.

$$L(a, b) = \int \cdots \int_Q f(u_1, \ldots, u_n) du_1 \ldots du_n - 2 \int \cdots \int_D f(u_1, \ldots, u_n) du_1 \ldots du_n \approx \Delta \left((1 + \alpha)^n 1^T f_Q - 2 1^T f_D\right)$$
$$\nabla_a L \approx 2 \Delta \text{diag}^{-1}(r) \left(2Mf_D - \overline{M}_Q f_Q\right)$$
$$\nabla_b L \approx 2 \Delta \text{diag}^{-1}(r) \left(-2Mf_D + \overline{M}_Q f_Q\right)$$

(C.30)

where $\overline{M}_Q$ is the outer region mask defined as follows.

$$\overline{M}_Q = (1 + \alpha)^{n-1} \left((1 + \frac{\alpha}{2})M + \frac{\alpha}{2} \overline{M}\right)$$
$$M_Q = (1 + \alpha)^{n-1} \left((1 + \frac{\alpha}{2})M + \frac{\alpha}{2} \overline{M}\right)$$

(C.31)

**White-Box OIR** (OIR\_W.) The following formulation is white-box in nature (it
needs the gradient of the function \( f \) in order to update the current estimates of the bound. This is useful when the function in hand is differentiable (e.g., DNN), to obtain more intelligent regions, rather than the regions surrounded by near 0 values of the function \( f \). We set \( \lambda = \frac{\alpha}{\beta} \) in Eq (C.17), where \( \alpha \) is the small boundary factor of the outer area, \( \beta \) is the emphasis factor (we will show later how it determines the emphasis on the function vs the gradient). Hence, the objective in Eq (C.17) becomes:

\[
\arg \min_{a,b} L = \arg \min_{a,b} \text{Area}_{\text{out}} - \lambda \text{Area}_{\text{in}}
\]

\[
= \arg \min_{a,b} \int_{a}^{b} f(u)du + \int_{b}^{B} f(u)du - \frac{\alpha}{\beta} \int_{a}^{b} f(u)du
\]

\[
= \arg \min_{a,b} \frac{\beta}{\alpha} \int_{a-\alpha \frac{b-a}{2}}^{b+\alpha \frac{b-a}{2}} f(u)du - (1 + \frac{\beta}{\alpha}) \int_{a}^{b} f(u)du
\]

(C.32)

using Libeniz rule as in Lemma 1, we get the following derivatives of the bound \( a \):

\[
\frac{\partial L}{\partial a} = \frac{\beta}{\alpha} \left( f(a) - f\left(a - \alpha \frac{b-a}{2}\right)\right)
\]

\[
- \frac{\beta}{2} f\left(b + \alpha \frac{b-a}{2}\right) - \frac{\beta}{2} f\left(a - \alpha \frac{b-a}{2}\right) + f(a)
\]

(C.33)

now since \( \lambda^* \) should be small for the optimal objective, then as \( \lambda \to 0, \alpha \to 0 \) and hence the derivative in Eq (C.33) becomes the following.

\[
\lim_{\alpha \to 0} \frac{\partial L}{\partial a} = \lim_{\alpha \to 0} \beta \frac{f(a) - f\left(a - \alpha \frac{b-a}{2}\right)}{\alpha}
\]

\[
- \frac{\beta}{2} f(b) - \frac{\beta}{2} f(a) + f(a)
\]

(C.34)

We can see that the first term is proportional to the derivative of \( f \) at \( a \), and hence the expression becomes:

\[
\lim_{\alpha \to 0} \frac{\partial L}{\partial a} = \frac{\beta}{2} \left( (b-a)f'(a) + f(b)\right) + (1 - \frac{\beta}{2})f(a)
\]

(C.35)
we can see that the update rule depends on the function value and the derivative of $f$ at the boundary $a$ with $\beta$ controlling the dependence. Similarly for the boundary $b$ we can see the following direction

$$\lim_{\alpha \to 0} \frac{\partial L}{\partial b} = \frac{\beta}{2} \left( (b-a)f'(b) + f(a) \right) - (1 - \frac{\beta}{2})f(b)$$  \hspace{1cm} (C.36)

If $\beta \to 0$, the update directions in Eq (C.35,C.36) collapse to the unregularized naive update in Eq (C.6).

**Extension to n-dimension.** Let's start by $n = 2$. Now, we have $f : \mathbb{R}^2 \to (0,1)$, and with the following constrains on the outer region.

$$A_1 = a_1 - \frac{b_1 - a_1}{2}, \quad B_1 = b_1 + \frac{b_1 - a_1}{2}$$
$$A_2 = a_2 - \frac{b_2 - a_2}{2}, \quad B_1 = b_2 + \frac{b_2 - a_2}{2}$$  \hspace{1cm} (C.37)

we follow similar approach as in Eq (C.32) to obtain the following expression.

$$L(a_1, a_2, b_1, b_2) =$$
$$\frac{\beta}{\alpha} \int_{A_2}^{B_2} \int_{A_1}^{B_1} f(u,v) dv du - (1 + \frac{\beta}{\alpha}) \int_{a_2}^{b_2} \int_{a_1}^{b_1} f(u,v) dv du$$  \hspace{1cm} (C.38)

We apply the trapezoidal approximation on the loss to obtain the following approximation.

$$L(a_1, a_2, b_1, b_2) \approx -1 + (1 + \alpha)^2$$
$$\left( f(A_1, A_2) + f(B_1, A_2) + f(A_1, B_2) + f(B_1, B_2) \right)$$
$$\left( f(a_1, a_2) + f(b_1, a_2) + f(a_1, b_2) + f(b_1, b_2) \right)$$  \hspace{1cm} (C.39)

to find the update direction for the first bound $a_1$ by taking the partial derivative of the function in Eq (C.21) we get the following update direction for $a_1$ along with its
trapezoidal approximation in order to able to compute it during the optimization:

\[
\frac{\partial L}{\partial a_1} = -\frac{\beta}{\alpha}(1 + \frac{\alpha}{2}) \int_{A_2}^{B_2} \left( f(A_1, v) + \frac{\alpha}{2} f(B_1, v) \right) dv \\
+ (1 + \frac{\beta}{\alpha}) \int_{a_2}^{b_2} f(a_1, v) dv \\
\approx \frac{(b_2 - a_2)}{2} ( \\
- (1 + \alpha)( \frac{\beta}{\alpha} + \frac{\beta}{2})(f(A_1, A_2) + f(A_1, B_2)) \\
+ \frac{\beta}{2} (f(B_1, A_2) + f(B_1, B_2)) ) \\
+ (1 + \frac{\beta}{\alpha})( f(a_1, a_2) + f(a_1, b_2) )
\]

(C.40)

grouping the terms which are divided by \(\alpha\) together and then taking the limit of \(\alpha \rightarrow \infty\) (as explained in the 1-d case ), we get the following expressions.

\[
\lim_{\alpha \rightarrow 0} \frac{\partial L}{\partial a_1} \approx \frac{(b_2 - a_2)}{2} ( \\
- \beta( ( f(a_1, a_2) + f(a_1, b_2)) - (f(A_1, A_2) + f(A_1, B_2)) ) \\
- \lim_{\alpha \rightarrow 0} \frac{3\beta}{2} (f(A_1, A_2) + f(A_1, B_2) ) \\
- \lim_{\alpha \rightarrow 0} \frac{\beta}{2} (f(B_1, A_2) + f(B_1, B_2) ) \\
+ \left( f(a_1, a_2) + f(a_1, b_2) \right)
\]

(C.41)

Noting that the first term is related to the directional derivatives of \(f\), we get the following limit expression

\[
\lim_{\alpha \rightarrow 0} \frac{\partial L}{\partial a_1} \approx \frac{(b_2 - a_2)}{2} ( \\
\beta( \nabla f(a_1, a_2) \cdot \left(\frac{b_1-a_1}{b_2-a_2}\right) + \nabla f(a_1, b_2) \cdot \left(\frac{b_1-a_1}{b_2-a_2}\right) ) \\
+ \left(1 - \frac{3\beta}{2}\right)( f(a_1, a_2) + f(a_1, b_2) ) \\
- \frac{\beta}{2} (f(b_1, a_2) + f(b_1, b_2) )
\]

(C.42)

Doing similar steps to the first bound for the other three bounds we obtain the full
update directions for \((a_1, a_2, b_1, b_2)\)

\[
\lim_{\alpha \to 0} \frac{\partial L}{\partial a_1} \approx \frac{(b_2 - a_2)}{2} \\
\beta( \nabla f(a_1, a_2) \cdot \begin{pmatrix} b_1 - a_1 \\ b_2 - a_2 \end{pmatrix} + \nabla f(a_1, b_2) \cdot \begin{pmatrix} b_1 - a_1 \\ -(b_2 - a_2) \end{pmatrix} ) \\
+ \left(1 - \frac{3\beta}{2}\right) \left( f(a_1, a_2) + f(a_1, b_2) \right) \\
- \frac{\beta}{2} ( f(b_1, a_2) + f(b_1, b_2) )
\]

(C.43)

\[
\lim_{\alpha \to 0} \frac{\partial L}{\partial b_1} \approx \frac{(b_2 - a_2)}{2} \\
\beta( \nabla f(b_1, a_2) \cdot \begin{pmatrix} b_1 - a_1 \\ -(b_2 - a_2) \end{pmatrix} + \nabla f(b_1, b_2) \cdot \begin{pmatrix} b_1 - a_1 \\ (b_2 - a_2) \end{pmatrix} ) \\
- \left(1 - \frac{3\beta}{2}\right) \left( f(b_1, a_2) + f(b_1, b_2) \right) \\
+ \frac{\beta}{2} ( f(a_1, a_2) + f(a_1, b_2) )
\]

(C.44)

\[
\lim_{\alpha \to 0} \frac{\partial L}{\partial a_2} \approx \frac{(b_2 - a_2)}{2} \\
\beta( \nabla f(a_1, a_2) \cdot \begin{pmatrix} -(b_1 - a_1) \\ (b_2 - a_2) \end{pmatrix} + \nabla f(b_1, a_2) \cdot \begin{pmatrix} -(b_1 - a_1) \\ (b_2 - a_2) \end{pmatrix} ) \\
+ \left(1 - \frac{3\beta}{2}\right) \left( f(a_1, a_2) + f(b_1, a_2) \right) \\
- \frac{\beta}{2} ( f(a_1, b_2) + f(b_1, b_2) )
\]

(C.45)

\[
\lim_{\alpha \to 0} \frac{\partial L}{\partial b_2} \approx \frac{(b_2 - a_2)}{2} \\
\beta( \nabla f(a_1, b_2) \cdot \begin{pmatrix} -(b_1 - a_1) \\ -(b_2 - a_2) \end{pmatrix} + \nabla f(b_1, b_2) \cdot \begin{pmatrix} -(b_1 - a_1) \\ -(b_2 - a_2) \end{pmatrix} ) \\
- \left(1 - \frac{3\beta}{2}\right) \left( f(a_1, b_2) + f(b_1, b_2) \right) \\
+ \frac{\beta}{2} ( f(a_1, a_2) + f(b_1, a_2) )
\]

(C.46)

Now, for \( f : \mathbb{R}^n \to (0, 1)\) Following previous notations we have the following expressions
for the loss and update directions for the bound

\[ L(a, b) \approx \frac{(1 + \alpha)^{nTf_D}}{1^Tf_D} - 1 \]

\[ \nabla_a L \approx \Delta \left( \text{diag}^{-1}(r)\mathbf{M}_Df_D + \beta \text{diag}(\mathbf{MG}_D) + \beta \mathbf{s} \right) \tag{C.47} \]

\[ \nabla_b L \approx \Delta \left( -\text{diag}^{-1}(r)\mathbf{M}_Df_D + \beta \text{diag}(\mathbf{MG}_D) + \beta \mathbf{s} \right) \]

where the mask is the special mask

\[ \bar{\mathbf{M}}_D = \left( \gamma_n \mathbf{M} - \beta \mathbf{M} \right) \]

\[ \mathbf{M}_D = \left( \gamma_n \mathbf{M} - \beta \mathbf{M} \right) \tag{C.48} \]

\[ \gamma_n = 2 - \beta(2n - 1) \]

Where \( \text{diag}(.) \) is the diagonal matrix of the vector argument or the diagonal vector of the matrix argument. \( \mathbf{s} \) is a weighted sum of the gradient from other dominions \( (i \neq k) \) contributing to the update direction of dimension \( k \), where \( k \in \{1, 2, ..., n\} \).

\[ s_k = \frac{1}{r_k} \sum_{i=1,i \neq k}^{n} r_i \left( (\bar{\mathbf{M}}_{i,:} - \mathbf{M}_{i,:}) \odot \mathbf{M}_{k,:} \right) \mathbf{G}_{i,i} \]

\[ \bar{s}_k = \frac{1}{r_k} \sum_{i=1,i \neq k}^{n} r_i \left( (\mathbf{M}_{i,:} - \bar{\mathbf{M}}_{i,:}) \odot \mathbf{M}_{k,:} \right) \mathbf{G}_{i,i} \tag{C.49} \]

\[ k \in \{1, 2, ..., n\} \]

### C.2.5 Trapezoidal Approximation Formulation

Here we use the trapezoidal approximation of the integral, a first-order approximation from Newton-Cortes formulas for numerical integration [118]. The rule states that a definite integral can be approximated as follows:

\[ \int_{a}^{b} f(u)du \approx (b - a) \frac{f(a) + f(b)}{2} \tag{C.50} \]
asymptotic error estimate is given by \(-\frac{(b-a)^2}{48} \left[ f'(b) - f'(a) \right] + O(0.125)\). So as long the derivatives are bounded by some Lipschitz constant \(\mathbb{L}\), then the error becomes bounded by the following \(|\text{error}| \leq \mathbb{L}(b-a)^2\). The regularized loss of interest in Eq (C.5) becomes the following.

\[
L = -\text{Area}_{\text{in}} + \lambda |b-a|^2
\approx -(b-a) \frac{f(a) + f(b)}{2} + \lambda |b-a|^2
\] (C.51)

taking the derivative of \(L\) approximation directly with respect to these bounds yields the following update directions which are different from the expressions in Eq (C.6)

\[
\frac{\partial L}{\partial a} = -\frac{b-a}{2} f'(a) + \frac{f(a) + f(b)}{2} - \lambda (b-a)
\frac{\partial L}{\partial b} = -\frac{b-a}{2} f'(b) - \frac{f(a) + f(b)}{2} + \lambda (b-a)
\] (C.52)

note that it needs the first derivative \(f'(.)\) of the function \(f\) evaluated at the bound to update that bound.

**Extension to n-dimensions.**  
Let's start with \(n = 2\). Now, we have \(f : \mathbb{R}^2 \rightarrow (0,1)\), and we define the loss integral as a function of four bounds of a rectangular region and apply the trapezoidal approximation as follows.

\[
L(a_1, a_2, b_1, b_2) = -\int_{a_1}^{b_1} \int_{a_2}^{b_2} f(u, v) dvdu
\quad + \frac{\lambda}{2} |b_1 - a_1|^2 + \frac{\lambda}{2} |b_2 - a_2|^2
\approx \frac{\lambda}{2} |b_1 - a_1|^2 + \frac{\lambda}{2} |b_2 - a_2|^2
\quad - \frac{(b_1 - a_1)(b_2 - a_2)}{4} \left( f(a_1, a_2) + f(b_1, a_2) + f(a_1, b_2) + f(b_1, b_2) \right)
\] (C.53)
Then following similar steps as in the one-dimensional case we can obtain the following update directions for the four bounds

\[
\frac{\partial L}{\partial a_1} = (b_1 - a_1)(b_2 - a_2) \frac{f_1'(a_1, a_2) + f_1'(a_1, b_2)}{4} - (b_2 - a_2) \frac{f(a_1, a_2) + f(a_1, b_2) + f(b_1, a_2) + f(b_1, b_2)}{4} + \lambda(b_1 - a_1) \\
+ (b_1 - a_1) \frac{f(a_1, a_2) + f(a_1, b_2) + f(b_1, a_2) + f(b_1, b_2)}{4} + \lambda(b_2 - a_2) \\
\frac{\partial L}{\partial b_1} = (b_1 - a_1)(b_2 - a_2) \frac{f_1'(a_1, a_2) + f_1'(b_1, a_2)}{4} - (b_2 - a_2) \frac{f(a_1, a_2) + f(a_1, b_2) + f(b_1, a_2) + f(b_1, b_2)}{4} + \lambda(b_1 - a_1) \\
\frac{\partial L}{\partial b_2} = (b_1 - a_1)(b_2 - a_2) \frac{f_2'(a_1, b_2) + f_2'(b_1, b_2)}{4} - (b_2 - a_2) \frac{f(a_1, a_2) + f(a_1, b_2) + f(b_1, a_2) + f(b_1, b_2)}{4} + \lambda(b_2 - a_2)
\]

Where \( f_1'(.) = \frac{\partial f(u,v)}{\partial u}, f_2'(.) = \frac{\partial f(u,v)}{\partial v} \). extending the 2-dimensional to general n-dimensions is straightforward. For \( f : \mathbb{R}^n \rightarrow (0, 1) \), we define the following. Let the left bound vector be \( a = [a_1, a_2, ..., a_n] \) and the right bound vector \( b = [b_1, b_2, ..., b_n] \) define the n-dimensional hyper-rectangle region of interest. The region is then defined as follows: \( D = \{ x : a \leq x \leq b \} \), Here, we assume the size of the region is positive at every dimension, i.e. \( r = b - a > 0 \). The volume of the region \( D \) normalized by exponent of dimension \( n \) is expressed as in Eq (C.11), the region \( D \) is defined as in Eq (C.12), \( M \) is defined as in Eq (C.13), and \( f_D \) is defined as in Eq (C.14). now we
can see that the loss integral in Eq (C.50) becomes as follows.

\[
L(a, b) = \int \cdots \int_D f(u_1, \ldots, u_n) \, du_1 \cdots du_n + \frac{\lambda}{2} |r|^2
\approx \Delta 1^T f_D + \frac{\lambda}{2} |r|^2
\]  

(C.58)

Similarly to Eq (C.14), we define the gradient matrix \(G_D\) as the matrix of all gradient vectors evaluated at all corner points of \(D\)

\[
G_D = \begin{bmatrix}
\nabla f(d^1) & \nabla f(d^2) & \cdots & \nabla f(d^{2^n})
\end{bmatrix}^T
\]  

(C.59)

The update directions for the left bound \(a\) and the right bound \(b\) becomes as follows by the trapezoid approximation

\[
\nabla_a L \approx \Delta \left( \text{diag}(\overline{M}G_D) + 1^T f_D \text{diag}^{-1}(r) 1 \right) + \lambda r
\]

\[
\nabla_b L \approx \Delta \left( \text{diag}(MG_D) - 1^T f_D \text{diag}^{-1}(r) 1 \right) - \lambda r
\]  

(C.60)

Where \(\overline{M}\) is the complement of the binary mask matrix \(M\).

C.3 Analysis

C.3.1 Detected Robust Regions

We apply the algorithms on two semantic parameters (the azimuth angle of the camera around the object, and the elevation angle around the object) that are regularly used in the literature [100, 28]. When we use one parameter (the azimuth), we fix the elevation to 35°. We used two instead of larger numbers because it is easier to verify and visualize 2D, unlike higher dimensions. Also, the complexity of sampling increases exponentially with dimensionality for those algorithms (albeit being much better than grid sampling, see Table 4.2). The regions in Figure C.3 look like vertical rectangles most of the time. This is because the scale of the horizontal axis (0,360) is
much smaller than the vertical axis (-10,90), so most regions are squares but look like rectangles because of figure scales.

### C.3.2 Hyper-Parameters

How to select the hyperparameters in all the above algorithms? The answer is not straightforward. For the $\lambda$ in the naive approach, it is set experimentally by trying different values and using the one which detects some regions that are known to behave robustly. The values we found for this are $\lambda = 0.07 \sim 0.1$. The learning rate $\eta$ is easier to detect by observing the convergence and depends on the full range of study. A rule of thumb is to make 0.001 of the full range. For the OIR formulations, we have the boundary factor $\alpha$ which we set to 0.05. A rule of thumb is to set it to be between $0.5 \sim 1/N$, where $N$ is the number of samples needed for that dimension to adequately characterize the space. In our case $N = 180$, so $1/180 \approx 0.005$. The only hyperparameter with a mathematically established bound is the emphasis factor of the OIR\_W formulation $\beta$. The bound shown in Table 4.2 which is $0 \leq \beta \leq \frac{2}{2n-1}$ can be shown as follows. We start from Eq (C.48). As we can see, the most important term is $\gamma_n$. It dictates how the function at the boundaries determines the next move of the bounds. Here $\gamma_n$ should always be positive to ensure the correct direction of the movement for positive function evaluation.

$$
\begin{align*}
\gamma_n &> 0 \\
2 - (2n - 1)\beta &> 0 \\
\beta &< \frac{2}{2n - 2}
\end{align*}
$$

### C.3.3 Dataset

The dataset used is collected from ShapeNet [16] and consists of 10 classes and 10 shapes each that are all identified by at least ResNet50 trained on ImageNet. This
criterion is important to obtain valid NSM and DSM. The classes are ['aeroplane', 'bathtub', 'bench', 'bottle', 'chair', 'cup', 'piano', 'rifle', 'vase', 'toilet']. 
Figure C.1: **1D Network Semantic Maps NSM**: visualizing 1D Semantic Robustness profile for different networks averaged over 10 different shapes.
Figure C.2: **2D Network Semantic Maps NSM**: visualizing 2D Semantic Robustness profile for different networks averaged over 10 different shapes. Every row is a different class.
Figure C.3: **Qualitative Examples of Robust Regions**: visualizing different runs of the algorithm to find robust regions along with different renderings from inside these regions for those specific shapes used in the experiments.
Figure C.4: **Data Semantic Maps DSM**: visualizing Semantic Data Bias in the common training dataset *(i.e. ImageNet [12])* By averaging the Networks Semantic Maps (NSM) of different networks and on different shapes, Different classes have a different semantic bias in ImageNet as clearly shown in the maps above. The symmetry in the maps is attributed to the 3D symmetry of the objects.
D Appendix D

D.1 Empirical Justification for BBGAN

We want to show that as the size of the induced set \(|S_{u'}| \to \infty\), learning an adversary according to the BBGAN objective in Eq (D.1) converges to the fooling distribution of semantic parameters \(P_{u'}\) defined in Eq (D.2).

\[
\min_{G_u} \max_{D_u} L_{BBGAN}(G_u, D_u, S_u') = \\
E_{u \sim S_u'}[\log D_u(u)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]
\]  

\[(D.1)\]

\[u' \sim P_{u'} \iff Q(A, E_{u'}) \leq \epsilon; \quad u' \in [u_{\text{min}}, u_{\text{max}}]^d\]  

\[(D.2)\]

Here, the agent \(A\) interacts with the environment \(E_u\) and receives a score \(Q\) for a given parameter \(u\). The fooling threshold is \(\epsilon\).

D.1.1 Empirical Proof

We use the definition of Exhaustive Search (Algorithm 3.1) from the Audet and Hare book on derivative-free and black box optimization [156]. In this algorithm, we try to optimize an objective \(f : \mathbb{R}^d \to \mathbb{R}\) defined on a closed continuous global set \(\Omega\) by densely sampling a countable subset \(S = \{u_1, u_2, \ldots, u_N\} \subset \Omega\). Theorem 3.1 [156] states that as long as the exhaustive search continues infinitely from the set \(S\), the global solutions of \(f\) can be reached. Let’s assume the global solutions \(u^*\) of \(f\) exist and are defined as follows.
\[ u^* = \arg \min_u f(u) \quad s.t. \ u \in \Omega \quad (D.3) \]

Let’s denote the best solution reached up to the sample \( u_N \) to be \( u^\text{best}_N \). If the set \( S_{u^*} \) is the set of all global solutions \( u^* \), then Theorem 3.1 [156] states that

\[ u^\text{best}_N \in S_{u^*} = \{ u^* \} \quad \text{as} \ N \to \infty \quad (D.4) \]

Now let \( f(u) = \max(0, \ Q(E_u, A) - \epsilon) \) and let \( \Omega = [u_{\min}, u_{\max}] \), then the global solutions of the optimization:

\[ u^* = \arg \min_u \max(0, \ Q(E_u, A) - \epsilon) \quad \quad (D.5) \]

\[ s.t. \ u \in [u_{\min}, u_{\max}] \]

satisfy the two conditions in Eq (D.2) as follows.

\[ Q(A, E_{u^*}) \leq \epsilon; \quad u^* \in [u_{\min}, u_{\max}]^d \quad (D.6) \]

Hence, the set of all global solutions includes all the points in the fooling distribution:

\[ S_{u^*} = \{ u^* \} = \{ u' : u' \sim P_{u^*} \} \quad (D.7) \]

Therefore, as the sampling set size \( |S| \to \infty \), all the points \( u \) that lead to \( Q(E_u, A) \leq \epsilon \), achieve the minimum objective in Eq (D.5) of zero and the set of best observed values \( |\{ u^\text{best}_N \}| \to \infty \). This set is what we refer to as the induced set \( S_{u^*} \).

From Eq (D.4) and Eq (D.7), we can infer that the induced set will include all fooling parameters as follows.

\[ \text{as} \ N \to \infty, \quad S_{u^*} \to \{ u' : u' \sim P_{u^*} \} \quad (D.8) \]

Finally, if the set \( S_{u^*} \) has converged to the distribution \( P_{u^*} \) and we use \( S_{u^*} \) to train
the BBGAN in Eq (D.1), then according to proposition 2 from the original GAN paper by Goodfellow et al. [38], the adversary $G_u$ has learned the distribution $P_{u'}$ and hence satisfies the following equation:

$$\arg\min_{G_u} E_{u \sim G_u} [Q(A, E_u)]$$

s.t. \( \{u : u \sim G_u\} = \{u' : u' \sim P_{u'}\} \) \hspace{1cm} (D.9)

This concludes our empirical proof for our BBGAN.

D.2 Special Cases of Generic Adversarial attacks:

In Table D.1, we summarize the substitutions in the generic adversarial attack to get different special cases of adversarial attacks. In summary, the generic adversarial attack allows for static agents (like classifiers and detectors) and dynamic agents (like autonomous agents acting in a dynamic environment). It also covers the direct attack case of pixel perturbation on images to attack classifiers, as well as semantic attacks that try to fool the agent in a more realistic scenario (e.g. camera direction to fool a detector). The generic attack also allows for a more flexible way to define the attack success based on an application-specific threshold $\epsilon$ and the agent score $Q$. In the following, we provide a detailed explanation and mathematical meaning of the substitutions.

D.2.1 Pixel Adversarial Attack on Image Classifiers

Probably the most popular adversarial attack in the literature is a pixel-level perturbation to fool an image classifier. This attack can be thought of as a special case of our general formulation. In this case, the agent $A$ is a classifier $C : [0, 1]^n \rightarrow [l_1, l_2, ..., l_K]$ and the environment $E_u$ is a dataset containing the set $\Phi$ of all images in the classification dataset along with their respective ground truth labels, i.e. $\{(x_i, y_i)\}_{i=1}^{\mid\Phi\mid}$ and $y_i \in \{1, \ldots, K\}$. The softmax value of the true class is given by $l_{y_i} = \max C(x_i)$. 
### Substitutions of Special Cases of Adversarial Attacks

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent ( A )</td>
<td>( K )-class classifier ( C : [0,1]^n \rightarrow [l_1,l_2,...,l_K] ) ( l_j ) : the softmax value for class ( j )</td>
<td>( K )-class object detector ( F : [0,1]^n \rightarrow \mathbb{R}^{K \times N} ) ( N ) : number of detected objects</td>
<td>self-driving policy agent ( A ) e.g. network to regress controls</td>
</tr>
<tr>
<td>Parameters ( u )</td>
<td>the pixels noise added on attacked image</td>
<td>some semantic parameters describing the scene e.g. camera pose, object, light</td>
<td>some semantic parameters involved in the simulation e.g. road shape, weather, camera</td>
</tr>
<tr>
<td>Size ( d )</td>
<td>( n ) : the image dimension ( n = h \times w \times c )</td>
<td>number of semantic parameters parameterizing ( E_u )</td>
<td>number of semantic parameters parameterizing ( E_u )</td>
</tr>
<tr>
<td>Environment ( E_u )</td>
<td>dataset ( \Phi ) containing all images and their true class label ( \Phi = { (x_i,y_i) }_{i=1}^{i} )</td>
<td>dataset ( \Phi ) containing all images and their true class label ( \Phi = { (x_i,y_i) }_{i=1}^{i} )</td>
<td>simulation environment partially described by ( u ) that ( A ) navigates in to reach</td>
</tr>
<tr>
<td>Range ( [a_{min}, a_{max}] )</td>
<td>( [x_{min}, x_{max}] ) are the min and max of each pixel value in the image ( x )</td>
<td>( [−1,1]^d )</td>
<td>( [−1,1]^d )</td>
</tr>
<tr>
<td>Observation ( o_t )</td>
<td>the attacked image after adding fooling noise</td>
<td>the rendered image using the scene parameters ( u )</td>
<td>sequence of rendered images ( A ) observes during the simulation episode</td>
</tr>
<tr>
<td>Episode Size ( T )</td>
<td>1</td>
<td>1</td>
<td>1: step when success reached</td>
</tr>
<tr>
<td>Time step ( t )</td>
<td>1</td>
<td>1</td>
<td>time step where agent ( A ) has to decide on action ( a_t ) given what it has observed ( o_t )</td>
</tr>
<tr>
<td>Fooling Threshold ( \epsilon )</td>
<td>fixed small fraction of 1 e.g. = 0.3</td>
<td>fixed small fraction of 1 e.g. = 0.6</td>
<td>sequence of rendered images ( A ) observes during the simulation episode</td>
</tr>
<tr>
<td>Observation ( o_t )</td>
<td>attacked image after added noise = ( x_i + u ), where ( x, u \in \mathbb{R}^n )</td>
<td>the rendered image using the scene parameters ( u )</td>
<td>sequence of rendered images ( A ) observes during the simulation episode</td>
</tr>
<tr>
<td>Agent Actions ( a_t(o_t) )</td>
<td>predicted softmax vector of attacked image</td>
<td>predicted confidence of the true class label</td>
<td>steering command to move the car/UAV in the next step</td>
</tr>
<tr>
<td>Reward Functions ( R(s_t, a_t) )</td>
<td>the difference between true and predicted softmax</td>
<td>predicted confidence of the true class label</td>
<td>1 : if success state reached 0: if success not reached</td>
</tr>
<tr>
<td>Score ( Q(A, E_u) )</td>
<td>the difference between true and predicted softmax</td>
<td>predicted confidence of the true class label</td>
<td>the average sum of rewards over five different episodes</td>
</tr>
</tbody>
</table>

**Table D.1: Special Cases of Generic Adversarial Attacks:** we summarize all the variable substitutions to get common adversarial attacks.

Parameter \( u \) defines the fooling noise to be added to the images \( i.e. d = n \). The observation is simply an image from \( \Phi \) with noise added: \( o_t = x_i + u \) for some \( i \in \{1,2,...,|\Phi|\} \). In classification, the environment is static with \( T = 1 \). To ensure the resulting image is in the admissible range, the noise added \( u \) should fall in the range \( [−x_{i,min}, 1−x_{i,max}] \), where \( x_{i,min}, x_{i,max} \) are the min and max pixel value for the image \( x_i \). The sole action \( a_1 \) is simply the softmax score of the highest scoring class la-
The reward function is $R(s_1, a_1) = Q(C, \Phi) = \max(l_{y_i} - l_j, 0)$. Here, $\epsilon = 0$ for the classifier fooling to occur, which means fooling occurs if $l_{y_i} - l_j \leq 0$. Using these substitutions in the hard constraint in Eq (D.2) translates to the following constraints on the perturbed image.

$$l_{y_i} \leq l_j , \ x_i + u \in [0, 1]^n \quad (D.10)$$

For a single image attack, we can rewrite Eq (D.10) as follows:

$$\max C(x) \leq \max_{j \neq y} C(x') , \ x' \in [0, 1]^n \quad (D.11)$$

We observe that the constraints in Eq (D.11) become the following constraints of the original adversarial pixel attack formulation on a classifier $C$.

$$\min_{x' \in [0, 1]^n} d(x, x') \ \text{s.t.} \ \arg \max C(x) \neq \arg \max C(x') \quad (D.12)$$

### D.2.2 Semantic Adversarial Attack on Object Detectors

Extending adversarial attacks from classifiers to object detectors is straightforward. We follow previous work [142] in defining the object detector as a function $F : [0, 1]^n \rightarrow (\mathbb{R}^{N \times K}, \mathbb{R}^{N \times 4})$, which takes an $n$-dimensional image as input and outputs $N$ detected objects. Each detected object has a probability distribution over $K$ class labels and a 4-dimensional bounding box for the detected object. We take the top $J$ proposals according to their confidence and discard the others. Analyzing the detector in our general setup is similar to the classifier case. The environment $E_u$ is static (i.e. $T = 1$), and it contains all images with ground truth detections. For simplicity, we consider one object of interest per image (indexed by $i$). The observation in this case is a rendered image of an instance of object class $i$, where the
environment parameter $u$ determines the 3D scene and how the image is rendered (e.g. the camera position/viewpoint, lighting directions, textures, etc.). Here, the observation is defined as the rendering function $o_1: [u_{\text{min}}, u_{\text{max}}]^d \to \mathbb{R}^n$. We use Blender [13] to render the 3D scene containing the object and to determine its ground truth bounding box location in the rendered image. The action $a_1$ by the agent/detector is simply the highest confidence score $l_i$ corresponding to class $i$ from the top $J$ detected boxes in $o_1$. The final score of $F$ is $Q(F, E_u) = l_i$. The attack on $F$ is considered successful, if $l_i \leq \epsilon$.

D.2.3 Semantic Adversarial Attack on Autonomous Agents

The semantic adversarial attack of an autonomous agent can also be represented in the general formulation of Algorithm 1 in the paper. Here, $A$ corresponds to the navigation policy, which interacts with a parametrized environment $E_u$. The environment parameter $u \in \mathbb{R}^d$ comprises $d$ variables describing the weather/road conditions, camera pose, environment layout etc. In this case, an observation $o_t$ is an image as seen from the camera view of the agent at time $t$. The action $a_t$ produced by the navigation policy is the set of control commands (e.g. gas and steering for a car or throttle, pitch, roll and yaw for a UAV). The reward function $R(s_t, a_t)$ measures if the agent successfully completes its task (e.g. 1 if it safely reaches the target position at time $t$ and 0 otherwise). The episode ends when either the agent completes its task or the maximum number of iterations $T$ is exceeded. Since the reward is binary, the $Q$ score is the average reward over a certain number of runs (five in our case). This leads to a fractional score $0 \leq Q \leq 1$. 
D.3 Boosting Strategy for BBGAN

D.3.1 Intuition for Boosting

Inspired by the classical Adaboost meta-algorithm [157], we use a boosting strategy to improve the performance of our BBGAN trained in Section 5.3 of Chapter 4. The boosting strategy of BBGAN is simply utilizing the information learned from one BBGAN by another BBGAN in a sequential manner. The intuition is that the main computational burden in training the BBGAN is not the GAN training but computing the agent \( A \) episodes (which can take multiple hours per episode in the case of self-driving experiments).

D.3.2 Description of Boosting for BBGANs

We propose to utilize the generator to generate samples that can be used by the next BBGAN. We start by creating the set \( \Omega_0 \) of the first stage adversary \( G_0 \). We then simply add the generated parameters \( u \) along with their computed scores \( Q \) to the training data of the next stage BBGAN (i.e. BBGAN-1). We start the next stage by inducing a new induced set \( S_u^1 \) (that may include part or all the previous stage induced set \( S_u^0 \)). However, the aim is to put more emphasis on samples that were generated in the previous stage. Hence, the inducer in the next stage can just randomly sample \( N \) points, compute their \( Q \) scores and add \( \beta \times N \) generated samples from BBGAN-0 to the \( N \) random samples. The entire set is then sorted based on the \( Q \) scores, where the lowest-scoring \( s_1 \) points that satisfy Eq (D.2) are picked as the induced set \( S_u^1 \). \( s_1 = |S_u^1| \). The BBGAN-1 is then trained according to Eq (D.1). Here \( \beta \) is the boosting rate of our boosting strategy which dictates how much emphasis is put on the previous stage (exploitation ratio) when training the next stage. The whole boosting strategy can be repeated more than once. The global set \( \Omega \) of all \( N \) sampled points and the update rule from one stage to another is described
**Algorithm 4:** Boosting Strategy for BBGAN

- **Requires:** environment $E_u$, Agent $A$, number of boasting stages $K$, boosting rate $\beta$, initial training size $N$
- Sample $N$ points to form $\Omega_0$ like in Eq (D.13)
- induce $S^0_u$ from $\Omega_0$
- learn adversary $G_0$ according to Eq (D.1)
- for $i \leftarrow 1$ to $K$
  - update boosted training set $\Omega_i$ from $\Omega_{i-1}$ as in Eq (D.14)
  - obtain $S^i_u$ from $\Omega_i$
  - train adversary $G_i$ as in Eq (D.1)
- **Returns:** last adversary $G_K$

by the following two equations:

\[
\Omega_0 = \{u_j \sim \text{Uniform}(u_{\text{min}}, u_{\text{max}})\}_{j=1}^N \tag{D.13}
\]

\[
\Omega_k = \Omega_{k-1} \cup \{u_j \sim G_{k-1}\}_{j=1}^{\lfloor \beta N \rfloor} \tag{D.14}
\]

These global sets $\Omega_k$ constitute the basis from which the inducer produces the induced set $S^k_u$. The adversary $G_k$ of boosting stage $k$ uses this induced set when training according to the BBGAN objective in Eq (D.1). Algorithm 4 summarizes the boosting meta-algorithm for BBGAN.

### D.3.3 Empirical proof for BBGAN Boosting

Here we want to show the effectiveness of boosting (Algorithm 4) on improving the performance of BBGAN from one stage to another. Explicitly, we want to show that the following statement holds, under some conditions.

\[
\mathbb{E}_{u_k \sim G_k}[Q(A, E_{u_k})] \leq \mathbb{E}_{u_{k-1} \sim G_{k-1}}[Q(A, E_{u_{k-1}})] \tag{D.15}
\]

This statement says that the expected score $Q$ of the sampled parameters $u_k$ from the adversary $G_k$ of stage $(k)$ BBGAN is bounded above by the score of the previous
stage, which indicates iterative improvement of the fooling adversary $G_k$ by lowering the score of the agent $A$ and hence achieving a better objective at the following relaxation of Eq (D.7).

$$\arg\min_{G} \mathbb{E}_{u \sim G}[Q(A, E_u)]$$ (D.16)

s.t. $\{u : u \sim G\} \subset \{u' : u' \sim P_{u'}\}$

Proof of Eq (D.15).

Let’s start by sampling random $N$ points as our initial $\Omega_0$ set as in Eq (D.13) and then learn BBGAN of the first stage and continue boosting as in Algorithm 4. Assume the following assumption holds,

$$|S^k_u| = \lfloor \beta N \rfloor, \forall k \in \{1, 2, 3, ...\}$$ (D.17)

then by comparing the average score $Q$ of the entire global set $\Omega_k$ at stage $k$ (denoted simply as $Q(\Omega_k)$) with the average score of the added boosting samples from the previous stage $\{u_j \sim G_{k-1}\}_{j=1}^{\lfloor \beta N \rfloor}$ as in Eq (D.14) (denoted simply as $Q(G_{k-1})$), two possibilities emerge:

1. Exploration possibility: $Q(\Omega_k) \leq Q(G_{k-1})$.

This possibility indicates that there is at least one new sample in the global set $\Omega_k$ that are not inherited from the previous stage adversary $G_{k-1}$, which is strictly better then $G_{k-1}$ samples with strictly lower $Q$ score. If the assumption in Eq (D.17) holds, then the induced set $S^k_u$ will include at least one new parameter that is not inherited from the previous stage and hence the average score of the induced set will be less than that of the generated by the previous stage.

$$Q(S^k_u) < Q(G_{k-1})$$ (D.18)
However, since the BBGAN of stage $k$ uses the induced set $S^k_u$ for training, we expect the samples to be correlated: $G_k \sim S^k_u$, and the scores to be similar as follows:

$$E_{u_k \sim G_k}[Q(A, E_{u_k})] = Q(S^k_u)$$  

(D.19)

Substituting Eq (D.19) in Eq (D.18) results in the inequality which makes the less strict inequality Eq (D.15) holds.

2. **Exploitation possibility:** $Q(\Omega_k) > Q(G_{k-1})$.

In this scenario, we don’t know for sure whether there is a new sample in $\Omega_k$ that is better than the inherited samples, but in the worst-case scenario, we will get no new sample with a lower score. In either case, the assumption in Eq (D.17) ensures that the new induced set $S^k_u$ is exactly the inherited samples from $G_{k-1}$ and the following holds.

$$Q(S^k_u) \leq Q(G_{k-1})$$  

(D.20)

Using the same argument as in Eq (D.19), we deduce that in this exploitation scenario Eq (D.15) is still satisfied. Hence, we prove that Eq (D.15) holds given the assumption in Eq (D.17).

**D.3.4 Experimental Details for Boosting**

We note that low $\beta$ values do not affect the training of our BBGAN since the induced set will generally be the same. Hence, we use $\beta = 0.5$, a high boosting rate. For practical reasons (computing 50% of the training data per boosting stage is expensive) we just compute 10% of the generated data and repeat it 5 times. This helps to stabilize the BBGAN training and forces it to focus more on samples that have low scores without having to evaluate the score function on 50% of the training data.
D.4 Detailed Results

Tables D.2 and D.3 show the detailed results for all three applications.

<table>
<thead>
<tr>
<th>Adversarial Attack Success Rate across Different Classes</th>
<th>airplane</th>
<th>bench</th>
<th>bike</th>
<th>boat</th>
<th>bottle</th>
<th>bus</th>
<th>car</th>
<th>chair</th>
<th>table</th>
<th>m.bike</th>
<th>train</th>
<th>truck</th>
<th>avg</th>
<th>µstd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Set</td>
<td>8.64</td>
<td>35.2</td>
<td>14.6</td>
<td>33.4</td>
<td>22.5</td>
<td>53.1</td>
<td>39.8</td>
<td>44.1</td>
<td>46.1</td>
<td>32.5</td>
<td>58.1</td>
<td>56.8</td>
<td>37.1</td>
<td>0.577</td>
</tr>
<tr>
<td>Random:</td>
<td>11.3</td>
<td>42.7</td>
<td>18.6</td>
<td>41.8</td>
<td>28.4</td>
<td>65.7</td>
<td>49.9</td>
<td>55.3</td>
<td>56.4</td>
<td>40.3</td>
<td>72.8</td>
<td>70.8</td>
<td>46.2</td>
<td>0.584</td>
</tr>
<tr>
<td>Multi-Class SVM</td>
<td>12.0</td>
<td>45.6</td>
<td>20.0</td>
<td>39.6</td>
<td>26.0</td>
<td>64.4</td>
<td>49.6</td>
<td>50.4</td>
<td>53.6</td>
<td>45.6</td>
<td>72.0</td>
<td>70.8</td>
<td>45.8</td>
<td>0.576</td>
</tr>
<tr>
<td>GP Regression</td>
<td>13.6</td>
<td>15.6</td>
<td>17.6</td>
<td>41.2</td>
<td>31.6</td>
<td>71.6</td>
<td>51.6</td>
<td>48.0</td>
<td>56.0</td>
<td>43.6</td>
<td>69.2</td>
<td>83.6</td>
<td>45.26</td>
<td>0.492</td>
</tr>
<tr>
<td>Gaussian</td>
<td>11.2</td>
<td>45.6</td>
<td>19.6</td>
<td>41.6</td>
<td>31.2</td>
<td>70.4</td>
<td>48.0</td>
<td>56.8</td>
<td>55.6</td>
<td>40.4</td>
<td>71.2</td>
<td>72.4</td>
<td>47.0</td>
<td>0.548</td>
</tr>
<tr>
<td>GMM10%</td>
<td>14.8</td>
<td>45.2</td>
<td>26.0</td>
<td>42.8</td>
<td>34.0</td>
<td>67.2</td>
<td>53.2</td>
<td>56.4</td>
<td>54.8</td>
<td>48.4</td>
<td>70.4</td>
<td>75.2</td>
<td>49.0</td>
<td>0.567</td>
</tr>
<tr>
<td>GMM50%</td>
<td>12.0</td>
<td>44.0</td>
<td>16.4</td>
<td>46.4</td>
<td>33.2</td>
<td>66.4</td>
<td>51.6</td>
<td>53.2</td>
<td>58.4</td>
<td>46.8</td>
<td>73.6</td>
<td>72</td>
<td>47.8</td>
<td>0.573</td>
</tr>
<tr>
<td>Bayesian</td>
<td>9.2</td>
<td>42.0</td>
<td>48.0</td>
<td>68.8</td>
<td>32.4</td>
<td>91.6</td>
<td>42.0</td>
<td>75.6</td>
<td>58.4</td>
<td>52.0</td>
<td>77.2</td>
<td>75.6</td>
<td>56.1</td>
<td>0.540</td>
</tr>
<tr>
<td>BBGAN (ours)</td>
<td>13.2</td>
<td>91.6</td>
<td>44.0</td>
<td>90.0</td>
<td>54.4</td>
<td>91.6</td>
<td>81.6</td>
<td>93.2</td>
<td>99.2</td>
<td>45.2</td>
<td>99.2</td>
<td>90.8</td>
<td>74.5</td>
<td>0.119</td>
</tr>
<tr>
<td>BBGAN (boost)</td>
<td>33.0</td>
<td>82.4</td>
<td>65.8</td>
<td>78.8</td>
<td>67.4</td>
<td>100</td>
<td>67.4</td>
<td>100</td>
<td>90.2</td>
<td>82.0</td>
<td>98.4</td>
<td>100</td>
<td>80.5</td>
<td>0.100</td>
</tr>
</tbody>
</table>

Table D.2: Attack Success Rate on Object Detection. We sample 250 parameters after the training phase of each model and sample a shape from the intended class. We then render an image according to these parameters and run the YOLOV3 detector to obtain a confidence score of the intended class. If this score $Q \leq \epsilon = 0.3$, then we consider the attack successful. The success rate is then recorded for that model, while $u_{std}$ (the mean of standard deviations of each parameter dimension) is recorded for each model.

<table>
<thead>
<tr>
<th>Straight One Curve Navigation 3 anchors 4 anchors 5 anchors</th>
<th>FR</th>
<th>µstd</th>
<th>FR</th>
<th>µstd</th>
<th>FR</th>
<th>µstd</th>
<th>FR</th>
<th>µstd</th>
<th>FR</th>
<th>µstd</th>
<th>FR</th>
<th>µstd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Set</td>
<td>10.6</td>
<td>0.198</td>
<td>19.5</td>
<td>0.596</td>
<td>46.3</td>
<td>0.604</td>
<td>17.0</td>
<td>0.607</td>
<td>23.5</td>
<td>0.544</td>
<td>15.8</td>
<td>0.578</td>
</tr>
<tr>
<td>Random</td>
<td>8.0</td>
<td>0.194</td>
<td>18.0</td>
<td>0.623</td>
<td>48.0</td>
<td>0.572</td>
<td>22.0</td>
<td>0.602</td>
<td>30.0</td>
<td>0.550</td>
<td>16.0</td>
<td>0.552</td>
</tr>
<tr>
<td>Multi-Class SVM</td>
<td>96.0</td>
<td>0.089</td>
<td>100</td>
<td>0.311</td>
<td>100</td>
<td>0.517</td>
<td>24.0</td>
<td>0.595</td>
<td>30.0</td>
<td>0.510</td>
<td>14.0</td>
<td>0.980</td>
</tr>
<tr>
<td>GP Regression</td>
<td>100</td>
<td>0.014</td>
<td>100</td>
<td>0.268</td>
<td>100</td>
<td>0.700</td>
<td>74.0</td>
<td>0.486</td>
<td>94.0</td>
<td>0.492</td>
<td>44.0</td>
<td>0.486</td>
</tr>
<tr>
<td>Gaussian</td>
<td>54.0</td>
<td>0.087</td>
<td>30.0</td>
<td>0.528</td>
<td>64.0</td>
<td>0.439</td>
<td>49.3</td>
<td>0.573</td>
<td>56.0</td>
<td>0.448</td>
<td>28.7</td>
<td>0.568</td>
</tr>
<tr>
<td>GMM10%</td>
<td>90.0</td>
<td>0.131</td>
<td>72.0</td>
<td>0.541</td>
<td>98.0</td>
<td>0.571</td>
<td>57.0</td>
<td>0.589</td>
<td>63.0</td>
<td>0.460</td>
<td>33.0</td>
<td>0.558</td>
</tr>
<tr>
<td>GMM50%</td>
<td>92.0</td>
<td>0.122</td>
<td>68.0</td>
<td>0.556</td>
<td>100</td>
<td>0.559</td>
<td>54.0</td>
<td>0.571</td>
<td>60.0</td>
<td>0.478</td>
<td>40.0</td>
<td>0.543</td>
</tr>
<tr>
<td>BBGAN (ours)</td>
<td>100</td>
<td>0.048</td>
<td>98.0</td>
<td>0.104</td>
<td>98.0</td>
<td>0.137</td>
<td>42.0</td>
<td>0.161</td>
<td>94.0</td>
<td>0.134</td>
<td>86.0</td>
<td>0.202</td>
</tr>
<tr>
<td>BBGAN (boost)</td>
<td>100</td>
<td>0.014</td>
<td>100</td>
<td>0.001</td>
<td>100</td>
<td>0.058</td>
<td>86.0</td>
<td>0.084</td>
<td>98.0</td>
<td>0.030</td>
<td>92.0</td>
<td>0.003</td>
</tr>
</tbody>
</table>

Table D.3: ASR for Autonomous Driving (CARLA) and UAV Racing Track Generation (Sim4CV). Each method produces 50 samples and we show the success rate and the mean of the standard deviation per parameter. We set the fooling threshold to 0.6 and 0.7 for autonomous driving and racing track generation respectively.

D.5 Qualitative Examples

Figure D.1 shows some qualitative examples for each of the 12 object classes. These adversarial images were rendered according to parameters generated by BBGAN.
Figure D.1: **BBGAN Qualitative Examples in Object Detection** - Some sample images for each class that were rendered according to parameters generated by BBGAN which fooled the object detector.
D.6 Training data

In the following we show some examples of the training data for each of the applications.

D.6.1 Object Detection

Object detection is one of the core perception tasks commonly used in autonomous navigation. Its goal is to determine the bounding box and class label of objects in an image. You Only Look Once (YOLO) object detectors popularized a single-stage approach, in which the detector observes the entire image and regresses the boundaries of the bounding boxes and the classes directly [158]. This trades off the accuracy of the detector for speed, making real-time object detection possible.

Agent. Based on its suitability for autonomous applications, we choose the very fast, state-of-the-art YOLOv3 object detector [123] as the agent in our SADA framework. It achieves a competitive mAP score on the MS-COCO detection benchmark and it can run in real-time [159].

Environment. We use Blender open-source software to construct a scene based on freely available 3D scenes and CAD models. The scene was picked to be an urban scene with an open area to allow for different rendering setups. The scene includes one object of interest, one camera, and one main light source all directed toward the center of the object. The light is a fixed-strength spotlight located at a fixed distance from the object. The material of each object is semi-metallic, which is common for the classes under consideration. The 3D collection consists of 100 shapes of 12 object classes (airplane, bench, bicycle, boat, bottle, bus, car, chair, dining table, motorbike, train, truck) from Pascal-3D [135] and ShapeNet [16]. At each iteration, one shape from the intended class is randomly picked and placed in the middle of the scene. Then, the Blender-rendered image is passed to YOLOV3 for detection. Please refer to Figure D.2 for some sample images of the object detection dataset.
Environment parameters. We use eight parameters that have been shown to affect
detection performance and frequently occur in real setups. The object is centered
in the virtual scene, and the camera circles around the object keeping the object in
the center of the rendered image. The parameters were normalized to $[-1, 1]^8$ before
using them for learning and testing.

D.6.2 Self-Driving

There is a lot of recent work in autonomous driving, especially in the fields of robotics
and computer vision [136, 137]. In general, complete driving systems are very complex
and difficult to analyze or simulate. By learning the underlying distribution of failure
cases, our work provides a safe way to analyze the robustness of such a complete
system. While our analysis is done in simulation only, we would like to highlight that
sim-to-real transfer is a very active research field nowadays [138, 139].

Agent. We use an autonomous driving agent (based on CIL [137]), which was
trained on the environment $E_u$ with default parameters. The driving policy was
trained end-to-end to predict car controls given an input image and is conditioned on
high-level commands (e.g. turn right at the next intersection) in order to facilitate
autonomous navigation.

Environment. We use the recent CARLA driving simulator [14], the most realistic
open-source urban driving simulator currently available. We consider the three
common tasks of driving in a straight line, completing one turn, and navigating
between two random points. The score is measured as the average success of five
pairs of start and end positions. Figure D.3 shows some images of the CARLA [14]
simulation environment used for the self-driving car experiments.

Environment parameters. Since experiments are time-consuming, we restrict
ourselves to three parameters, two of which pertain to the mounted camera viewpoint,
and the third controls the appearance of the environment by changing the weather
setting (e.g. ’clear noon’, ’clear sunset’, ’cloudy after rain’, etc.). As such, we construct an environment by randomly perturbing the position and rotation of the default camera along the z-axis and around the pitch axis respectively, and by picking one of the weather conditions. Intuitively, this helps measure the robustness of the driving policy to the camera position (e.g. deploying the same policy in a different vehicle) and to environmental conditions.

D.6.3 UAV Racing

In recent years, UAV (unmanned aerial vehicle) racing has emerged as a new sport where pilots compete in navigating small UAVs through race courses at high speeds. Since this is a very interesting research problem, it has also been picked up by the robotics and vision communities [140].

**Agent.** We use a fixed agent to autonomously fly through each course and measure its success as a percentage of gates passed [141]. If the next gate was not reached within 10 seconds, we reset the agent at the last gate. We also record the time needed to complete the course. The agent uses a perception network that produces waypoints from image input and a PID controller to produce low-level controls.

**Environment.** We use the general-purpose simulator for computer vision applications, Sim4CV [15]. Sim4CV is not only versatile but also photo-realistic and provides a suitable environment for UAV racing. Figure D.4 shows some images of the Sim4CV simulator used for the UAV racing application.

**Environment parameters.** We change the geometry of the race course environment. We define three different race track templates with 3-5 2D anchor points, respectively. These points describe a second-order B-spline and are perturbed to generate various race tracks populated by uniformly spaced gates. Figure D.5 shows some samples from the UAV datasets.
Figure D.2: **Training Data for YOLOV3 Experiment** - Some sample images from the dataset used for object detection with YOLOV3. Note that in the actual dataset, each object has a random color regardless of its class. For clarity, we uniformly color each class in this figure.
Figure D.3: **Environment for Self-driving** - Some samples of the CARLA [14] simulator environment.
Figure D.4: **Environment for UAV Racing** - Some samples of the Sim4CV [15] simulator environment.
Figure D.5: **Training Data for 5-Anchors UAV Racing** - Some sample tracks from the dataset with 5 control points.
E Accepted Papers

E.1 Accepted Papers


- **Abdullah Hamdi**, Silvio Giancola, Bernard Ghanem. MVTN: Multi-View Transformation Network for 3D Shape Recognition, Internation conference on computer vision 2021, ICCV’21. *(This work is presented in Chapter 3 of the dissertation.)*


- **Abdullah Hamdi**, Bernard Ghanem. Towards Analyzing Semantic Robustness of Deep Neural Networks, European Conference on Computer Vision Workshops 2020, ECCVW’20. *[Best Paper Award]* *(This work is presented in Chapter 4 of the dissertation.)*

- **Abdullah Hamdi**, Matthias Mueller, Bernard Ghanem. SADA: Semantic Adversarial Diagnostic Attacks for Autonomous Applications, AAAI Conference on Artificial Intelligence 2020, AAAI’20. *[Spotlight]* *(This work is presented in Chapter 5 of the dissertation.)*