



AFRL-RI-RS-TR-2022-075

EXPLORING VULNERABILITY AND ROBUSTNESS OF DEEP LEARNING SYSTEMS

EDMUND T. PRATT JR. SCHOOL OF ENGINEERING,
DUKE UNIVERSITY

MAY 2022

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2022-075 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /

NOAH J. MCDONALD
Work Unit Manager

/ S /

GREGORY J. HADYNSKI
Assistant Technical Advisor
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

1. REPORT DATE		2. REPORT TYPE		3. DATES COVERED	
MAY 2022		FINAL TECHNICAL REPORT		START DATE JANUARY 2018	END DATE JANUARY 2022
4. TITLE AND SUBTITLE EXPLORING VULNERABILITY AND ROBUSTNESS OF DEEP LEARNING SYSTEMS					
5a. CONTRACT NUMBER FA8750-18-2-0057		5b. GRANT NUMBER N/A		5c. PROGRAM ELEMENT NUMBER 63788F	
5d. PROJECT NUMBER		5e. TASK NUMBER		5f. WORK UNIT NUMBER R2FQ	
6. AUTHOR(S) Yiran Chen, Mathew Inkawhich, Jingyang Zhang, Randy Linderman					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Edmund T. Pratt Jr. School of Engineering, Electrical and Computer Engineering Department Duke University, Box 90291 Durham NC 27708				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RITB 525 Brooks Road Rome NY 13441-4505			10. SPONSOR/MONITOR'S ACRONYM(S) RI		11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-RI-RS-TR-2022-075
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Deep neural networks have enabled modern computer vision systems to reach new heights of performance on a variety of challenging tasks. Despite the accuracy and efficiency benefits, the highly parameterized non-linear nature of deep networks makes them very difficult to interpret and prone to failure in the presence of an adversary or anomalous data. This vulnerability makes the integration of these models into our real-world systems troubling. This project has two broad threads: (1) we explore the vulnerability of deep neural networks by developing state-of-the-art adversarial attacks, and (2) we improve the robustness of models in challenging operating environments such as in open-world target recognition and federating learning scenarios.					
15. SUBJECT TERMS Deep learning, neural networks, machine learning robustness, machine learning security					
16. SECURITY CLASSIFICATION OF:				17. LIMITATION OF ABSTRACT	
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	SAR		18. NUMBER OF PAGES 85
19a. NAME OF RESPONSIBLE PERSON NOAH J. MCDONALD				19b. PHONE NUMBER (Include area code) N/A	

TABLE OF CONTENTS

Section	Page
List of Figures.....	ii
List of Tables	iv
1.0 SUMMARY	1
2.0 INTRODUCTION.....	2
3.0 METHODS, ASSUMPTIONS, AND PROCEDURES	4
3.1. Enhancing and Understanding Adversarial Attack Effectiveness	4
3.1.1 Fast Generative Poisoning Attack.....	4
3.1.2 Activation Attack.....	10
3.1.3 Feature Distribution Attack.....	13
3.1.4 Multi-Layer Feature Distribution Attack.....	16
3.1.5 No Label Space Overlap Attack.....	17
3.1.6 Snooping Attacks on Deep Reinforcement Learning.....	21
3.2. Enhancing Robustness to Adversarial Attacks and Anomalous Data	23
3.2.1 Soteria: Provable Defense Against Privacy Leakage in Federated Learning.....	23
3.2.2 Adversarial Outlier Exposure for Reliable Operation of SAR-ATR Models.....	28
3.2.3 Improving Out-of-Distribution Detection by Learning from the Deployment Environment.....	32
4.0 RESULTS AND DISCUSSION	38
4.1. Enhancing and Understanding Adversarial Attack Effectiveness	38
4.1.1 Fast Generative Poisoning Attack.....	38
4.1.2 Activation Attack.....	43
4.1.3 Feature Distribution Attack.....	46
4.1.4 Multi-Layer Feature Distribution Attack.....	51
4.1.5 No Label Space Overlap Attack.....	54
4.1.6 Snooping Attacks on Deep Reinforcement Learning.....	58
4.2. Enhancing Robustness to Adversarial Attacks and Anomalous Data	59
4.2.1 Soteria: Provable Defense Against Privacy Leakage in Federated Learning.....	59
4.2.2 Adversarial Outlier Exposure for Reliable Operation of SAR-ATR Models.....	61
4.2.3 Improving Out-of-Distribution Detection by Learning from the Deployment Environment.....	64
5.0 CONCLUSIONS	71
6.0 REFERENCES.....	72
Appendix A – Publications.....	75
Appendix B – Abstract	76
7.0 List of Symbols, Abbreviations, and Acronyms.....	77

LIST OF FIGURES

Figure	Page
Figure 1. (Left) Numerical gradient method algorithm; (Right) Generative method – FastGPA algorithm	8
Figure 2. An overview of FastGPA	9
Figure 3. Illustration of Activation Attack. Given that the whitebox model (f_w) and blackbox model (f_b) are initially correct, the attack drives the layer L activations of the dog image towards the layer L activations of the plane. After the attack, the dog’s activations are similar to the plane’s and the perturbed image is classified as a plane to f_w and f_b	10
Figure 4. (Top) Given a Given a pre-trained whitebox model f , we capture the layer-wise and class-wise feature distributions with binary neural networks $g_{l,c}$, aiming to model the probability that the layer l features extracted from input x are from the class c feature distribution (i.e. $p(y = c f_{l(x)})$); (Bottom) Forward pass for FDA targeted attack	14
Figure 5. Visualization of forward pass construction to optimize our feature space attack objective	17
Figure 6. Overview of attack setup and procedure	19
Figure 7. Snooping Threat Model Overview	22
Figure 8. Illustration of the gradient updates of class-wise data in a batch	25
Figure 9. DLG attack results utilizing different parts of gradients	26
Figure 10. The local training process with our defense on a local device	28
Figure 11. Samples from both D_{id} and D_{ood}^{OE} sets.....	32
Figure 12. The trend of the loss and accuracy of the numerical gradient method under different group sizes for the MNIST dataset	39
Figure 13. The process of poisoned data generation under different configurations.....	40
Figure 14. Multi-sample poisoning attack on ImageNet	43
Figure 15. Transferability versus epsilon results for CIFAR-10 attacks transferring from different whitebox models to a ResNet-50 blackbox model.	44
Figure 16. Error, uTR, tSuc, and tTR rates versus depth for multiple transfer scenarios; The top two rows are transfers from a DN121 whitebox model and the bottom two rows are transfers from a VGG19bn whitebox model (Dataset: CIFAR-10).....	45
Figure 17. Targeted adversarial attack transfer results. The x-axis of each plot is the relative layer depth at which the adversarial example was generated from. Each row is a different whitebox model.....	46

Figure 18. Disruption versus layer depth for all transfer scenarios; Each row uses a different whitebox model; Each line is a different attack where all FDAs are $FDA+fd$	49
Figure 19. Correlation of a layer’s auxiliary models with the whitebox model output	50
Figure 20. Saliency maps of auxiliary models on several interesting inputs across model depth 51	
Figure 21. FDA versus ensemble attacks.....	53
Figure 22. Class correspondence matrices for the DINS Test 1 (top row) and DINS Test 2 (bottom row) environments.....	55
Figure 23. Performance reduction of DQN agents due to L^∞ and $L2$ bounded perturbations, the black dotted line represents a random-guess policy	59
Figure 24. Compared defenses on model accuracy and MSE between reconstructed image and raw image for different attack baselines and datasets; The pink vertical line is the boundary that the reconstructed image is unrecognizable by human eyes if MSE is higher.....	60
Figure 25. Comparing our defense with GC under the same pruning rate on model accuracy and MSE on a random image in CIFAR10.....	61
Figure 26. (Top 8 rows) Example images from each class of the MLRSNet dataset, organized by their partition into the MLRSNet-ID, MLRSNet-Hard and MLRSNet-Holdout splits; (Bottom 2 rows) Example images from some classes of the fMoW dataset which are used during outlier exposure training.....	65
Figure 27. Example images from the RSI-CB256, DIOR and Places365 datasets, which are used as additional OOD sets during testing in the EO scene classification environment.	66

LIST OF TABLES

Table	Page
Table 1. Average <i>cor</i> across 200 communication rounds under different settings.....	26
Table 2. The comparison of the numerical gradient method and FastGPA about computation time and accuracy under single-sample attack (NG = Numerical gradient method, GPA = FastGPA, im = improvement)	42
Table 3. Numerical transfer results (RN50 Blackbox)	45
Table 4. Transferability rates for 1000-class targeted attack tests using optimal layers	47
Table 5. ImageNet transfer results (notation = error / tSuc, $\epsilon = 16/255$).....	52
Table 6. Transfers in the DINS Test 1 environment (notation = error / tSuc).....	56
Table 7. DINS Test 2 transfers (notation = error / tSuc)	57
Table 8. ImageNet to Places365 transfers (notation = error / tSuc).....	58
Table 9. Detection results for different model training algorithms and combinations of K and J (Notation = AUROC / TNR@95TPR)	62
Table 10. Accuracy of K = 0 models for each training algorithm	63
Table 11. Main MLRSNet accuracy and detection results in a mixed environment (detection notation = AUROC / TNR@95TPR / AUPR).....	67
Table 12. Analyzing the impacts of ID and OOD library collection policy in different environmental conditions at different library sizes (Notation = AUROC-of-detecting-MLRSNet_Hard / AUROC-of-detecting-MLRSNet_Holdout).....	69

1.0 SUMMARY

The goal of this project is to bolster our understanding of Deep Neural Network (DNN) vulnerabilities and robustness characteristics in challenging scenarios. To do this, we focus on two broad, but independent threads. First, we study and develop a variety of adversarial machine learning techniques in a diverse range of machine learning tasks. Developing these adversarial techniques is beneficial as it allows us to better understand these complex models and they provide challenging benchmarks to test against as we continue to enhance robustness in the future. The second thread of our research deals with enhancing the robustness of DNNs in two challenging scenarios: federated learning frameworks, and open world target recognition environments. In total, this research resulted in nine publications that each advance the current state-of-the-art in their respective domains.

2.0 INTRODUCTION

Deep neural networks have enabled tremendous progress in the domain of machine learning and specifically computer vision. While the bulk of the recent research regarding these models seeks to improve accuracy and efficiency on hand-crafted benchmark tasks, the robustness of deep networks is not well understood. The highly parameterized nature of DNNs is both a blessing and a curse. On one hand, it enables performance levels that far exceed traditional machine learning models. On the other hand, DNNs are very difficult to interpret and fail to provide accurate notions of uncertainty. Therefore, it is important to continue to study and explore the vulnerabilities of these powerful models before integrating them into our most trusted systems.

The first major thread of our research is to explore the vulnerabilities of DNNs by crafting powerful adversarial attacks against a variety of models. Not only are adversarial attacks compelling from an offensive perspective, but they also serve as a tool to allow us to better understand and interpret complex model behaviors. Adversarial attacks also provide challenging robustness benchmarks that we can test against in the future. Our philosophy is that to create highly robust models, we must begin by attempting to fully understand all the ways that they can currently fail. Each of these works is individually motivated and explained in **Section 3.1**. In **Section 3.1.1**, we begin by discussing an early project on efficient model poisoning attacks that highlights a critical vulnerability of models with exposed training pipelines. Next, we cover a series of research projects that introduces and builds upon the novel idea of feature-space attacks. This class of attack is shown to be significantly more powerful than the existing output-space attacks in the more realistic blackbox attack settings. These papers are covered in **Sections 3.1.2 – 3.2.4**. In **Section 3.1.5**, we consider a previously unconsidered attack setting in which the blackbox target model shares no class distribution overlap with the target model. We show that we can use an adaptation of our feature distribution attacks to pose a significant threat to the blackbox models even in this challenging scenario. Finally, **Section 3.1.6** covers a novel class of blackbox adversarial attacks against reinforcement learning agents, which is a highly underexplored domain that is growing in popularity for control-based applications. Note that the experiments, results, and analysis for these projects are discussed in corresponding sections in **Section 4.0**.

The goal of our second thread of research involves directly enhancing the robustness of DNNs. As we detail in the first thread, adversarial attacks currently pose a significant risk to DNN-based systems. Before we trust these models enough to integrate into our most trusted systems, such as defense technologies, we must be sure that we account for all feasible forms of data corruption and variance. The first scenario that we consider, introduced in **Section 3.2.1**, involves formulating a principled defense against data inversion attacks in a distributed learning setting. Following this, in **Section 3.2.2** we significantly enhance the accuracy and robustness of Automatic Target Recognition (ATR) models operating in an open-world environment where we cannot guarantee that incoming data will contain classes inside the training distribution. In **Section 3.2.3** we

take this a step further and develop a memory-constrained online learning algorithm which enhances the robustness of ATR models in open-world environments by leveraging samples from the deployment environment. Again, the experiments, results, and discussion of these works are contained in corresponding sections in **Section 4.0**.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

In this section, we cover the motivations, methodologies, and procedures for each of our projects. **Section 3.1** covers the first thread of our research dealing with creating and understanding novel adversarial attack methods, whereas **Section 3.2** covers the second thread of research where we seek to improve the robustness of models to adversarial attack and anomalous data. The corresponding results, analysis and discussions related to these works are detailed in **Section 4.0**.

3.1. Enhancing and Understanding Adversarial Attack Effectiveness

3.1.1 Fast Generative Poisoning Attack.

Neural networks need to regularly collect data to perform re-training (fine-tuning) and make predictions, introducing at least two types of security issues—exploratory attack and causative attack. Exploratory attack does not change the parameters of the target model. Instead, the attacker fools the target model’s decisions by sending carefully crafted input samples. One recent example of exploratory attack is adversarial attack [1], which is a subset of evasion attack. The authors show that a DNN can be fooled by adding small perturbations onto its inputs though these perturbed inputs do not cause a large visual difference to humans. The neural network itself, however, does not change during an exploratory attack.

In contrast, causative attack denotes the situation when attackers can manipulate the training set and therefore change the parameters of the target model. One important type of causative attack is poisoning attack where artificially poisoned samples with attacking labels are sent into the target model. The target model will then gradually compromise during the model updating with these poisoned samples. Several schemes have been proposed to apply poisoning attack against support vector machines (SVMs) [2, 3]. However, very few works have been reported about the poisoning attacks against neural networks. One possible reason may be the training process of the neural network model, i.e., the weight updating during backpropagation, cannot be represented by a simple and explicit mathematical expression. As a result, applying the gradient-based poisoning attack involves complex second-order partial derivative which is not needed during the poisoning attack against SVMs. Besides the model-oriented poisoning attack, label flipping [2] is also a general and fast approach to generate poisoned samples: label flipping simply mislabels the training samples with incorrect classes to compromise the target model. However, label flipping is much less effective in attacking neural networks than our proposed FastGPA method due to its naïve nature.

Our objective is to explore how to degrade the testing accuracy of a neural network efficiently and effectively with maliciously poisoned samples. In this work, we first mathematically derive the gradients to maximize the attack efficiency, i.e., the accuracy

degradation of the target model, and examine the possibility of applying gradient-based numerical gradient method to generate the poisoned samples. We then propose a generative method, namely, FastGPA to increase the generation rate of the poisoned samples by bypassing the second order derivative calculation, which is the bottleneck in the numerical gradient method. The proposed FastGPA method can be summarized as follows: An autoencoder (generator) is used to generate the poisoned samples and send them to the target neural network model; The target model that receives the poisoned samples calculates the loss w.r.t. the normal samples and then sends the gradients back to the generator; The generator is then updated by the loss and the gradients.

Methodology. In this project, $x_i; \hat{x}_i; x_p \in \mathbb{R}^{\text{dim}}$ are used to denote the normal sample, the validation sample used to generate the poisoned sample, and the poisoned sample, respectively. $y_i; \hat{y}_i; y_p$ denote the corresponding labels, respectively. Superscripts (o) and (a) represent the variables of the original model and the attacked model, respectively. In this work, the entire parameter set of the original model, or the attacked model is expressed as $\mathbf{W}^{(o)}$ or $\mathbf{W}^{(a)}$, while $w_l^{(o)}$ and $w_l^{(a)}$ denote the parameters of only layer l . $f_{\mathbf{W}^{(a)}}(x_i)$ denotes the result of the last layer of $\mathbf{W}^{(a)}$ with input of x_i . $\varphi()$ is the activation function (e.g., a hyperbolic tangent (Tanh) or rectified linear unit (ReLU)).

As one type of causative attacks, poisoning attack allows the attacker to add artificial sample with arbitrary label into the training set. General poisoning attack assumes that the attacker has full knowledge of the target neural network model, including the network structure and the exact weight values, i.e., the white-box attack. This assumption is not realistic in real-world scenarios but may serve as a starting point. In this work, we will first explore the case under the whitebox setting and then extend to the gray-box setting where the attackers have access to the same dataset but not the NN model. We also choose a basic and extendable setting, i.e., the single-sample attack. Attackers can inject an arbitrary number of poisoned samples if their resources allow. We shall show that for small datasets, the single-sample attack is already very effective. To further meet the circumstance of a realistic scenario, we do not only consider the training dataset injected with poisoned samples but also split the attack into two separate phases, i.e., the model pre-training phase and the model re-training phase with injected poisoned samples.

Assume that a method of generating the poisoned samples is available, the poisoning attack process can be formulated as below:

- Step 1: Generate the poisoned sample x_p using the given method;
- Step 2: Inject x_p into the target model $\mathbf{W}^{(o)}$ to train an attacked model $\mathbf{W}^{(a)}$ (update once);
- Step 3: Send the validation sample x_i to $\mathbf{W}^{(a)}$ to get some useful information, e.g., the loss and the gradients at different iterations;
- Step 4: Based on the information obtained in Step 3, update the configurations and parameters of the method of generating poisoned samples;
- Step 5: Repeat the procedure from Step 1 until terminated, and the final poisoned sample x_p is obtained.

Gradient derivation of poisoned samples. The goal of poisoning attack is to minimize the testing accuracy of the target model $\mathbf{W}^{(o)}$ by injecting the poisoned sample x_p . Mathematically, this goal is as same as maximizing the loss of the model w.r.t. the validation sample x_i . When the target model adopts the *cross entropy* as its loss function, this goal can be expressed as:

$$x_p = \max_{x_p} L(x_p, \mathbf{W}^{(o)}), \quad (1)$$

$$L(x_p, \mathbf{W}^{(o)}) = - \sum_{i=1}^n \sum_{j=1}^{classes} 1\{y_i = j\} \log p(f_{\mathbf{W}^{(o)}(x_p)}(x_i) = j) \quad (2)$$

where n is the number of samples in the validation set. Equation 1 indicates that the maximum feasible solution in the loss space can be obtained by computing gradients w.r.t. x_p and adopting the gradient ascent procedure to generate x_p . In this subsection, we will show how to mathematically derive these gradients in neural networks. For illustration purpose, we focus on the single-sample poisoning attack. Then multi-sample attack can be achieved by conducting consecutive single-sample attacks.

First, we train a neural network on the existing training set and obtain $\mathbf{W}^{(o)}$. The corresponding loss $L_i^{(o)}$ w.r.t. x_i can be computed through a forward process on $\mathbf{W}^{(o)}$, such as $x_i \xrightarrow{\mathbf{W}^{(o)}} L_i^{(o)}$. Next, x_p with label y_p is injected. There are many ways to choose the initial x_p , e.g., sampling from a uniform distribution with a selected label or randomly choosing one from the normal samples with incorrect labels. The loss $L_p^{(o)}$ is then computed. We can update the network to $\mathbf{W}^{(a)}$ through backpropagation with the newly computed gradients, and obtain the loss $L_i^{(a)}$ as:

$$x_p \xrightarrow{\mathbf{W}^{(o)}} L_p^{(o)} \quad (3)$$

$$\mathbf{W}^{(a)} = \mathbf{W}^{(o)} - \alpha \nabla_{\mathbf{W}^{(o)}} L_p^{(o)}, \quad \text{and} \quad (4)$$

$$x_i \xrightarrow{\mathbf{W}^{(a)}} L_i^{(a)} \quad (5)$$

Finally, we need to compute $L_i^{(a)}$ and then update the poisoned sample along the direction of the gradient using a predetermined coefficient $x'_p = x_p + \alpha \nabla_{x_p} L_i^{(a)}$. Here $\nabla_{x_p} L_i^{(a)}$ can be expanded using the chain rule and expressed as the multiplication of multiple derivatives, such as:

$$\nabla_{x_p} L_{val}^{(a)} = \sum_{l=1}^{totalLayer} \frac{\partial L_{val}^{(a)}}{\partial w_l^{(a)}} \frac{\partial w_l^{(a)}}{\partial x_p}, \quad \text{where} \quad (6)$$

$$\frac{\partial L_{val}^{(a)}}{\partial w_l^{(a)}} = o_{l-1}^\top \delta_l \quad (7)$$

$$\frac{\partial w_l^{(a)}}{\partial x_p} = \frac{\partial \left(w_l^{(o)} - \alpha \frac{\partial L_p^{(o)}}{\partial w_l} \right)}{\partial x_p} = -\alpha \frac{\partial^2 L_p^{(o)}}{\partial w_l^{(o)} \partial x_p} \quad (8)$$

Here δ_l represents the sensitivity of the neurons in layer l to the loss. $o_l = \varphi(w_l o_{l-1})$ denotes the output of the neurons in layer l .

Numerical gradient method. The aforementioned mathematical study shows that updating the poisoned samples to a neural network involves the calculation of a second partial derivative in Equation 8. This term is difficult to be expressed as an explicit formula or computed by backpropagation since each of the two first derivatives is a variable of the other, due to the update of $\mathbf{W}^{(o)}$ to $\mathbf{W}^{(a)}$. Note that this is different from the calculation of the normal Hessian matrix.

In fact, rather than following Equations 6-8 we can compute $\nabla_{x_p} L_{val}^{(a)}$ directly based on the central difference:

$$\nabla_{x_p} L_{val}^{(a)} = \frac{L_{val}^{(a)}(x_p + \Delta x_p) - L_{val}^{(a)}(x_p - \Delta x_p)}{2\Delta x_p} \quad (9)$$

Here, Δx_p can be chosen as a small enough value. Specifically, we choose $\Delta x_p = 0.01 \times x_p$. Accordingly, Figure 1 (Left) summarizes the poisoned sample generation process of this *numerical gradient method*. The target model is assumed to be pre-trained with the complete training set, and the attacker is provided with a validation set to generate the poisoned sample x_p . The key to the proposed algorithm is to respectively compute the derivative w.r.t. each element of x_p , which will be summed up to generate the combined gradient $\nabla_{x_p} L_{val}^{(a)}$. x_p is then updated by multiplying the direction of the gradient, i.e., $\text{sign}(\nabla_{x_p} L_{val}^{(a)})$, with a rate factor α . We use the sign instead of its naive value is because the former one is more effective in real practices. α can be fixed at a predetermined value (e.g., 0.1) or dynamically changed in the process. However, this approach incurs multiple updates of $\mathbf{W}^{(o)}$ per iteration.

Algorithm 1 Numerical gradient method.

Input: Training set $X_{train} \subset \mathbb{R}^{dim}$ (dim is the dimension of the data), validation set X_{val} , attacking class y_p , generating rate α , loss threshold L_{th} .

Do:

1. Initialize and pre-train $W^{(o)}$ with X_{train} . Initialize $\{x_p(0), y_p\}$ pair. Set iteration $t = 0$ and $L_{val}^{(a)}(0) = 0$.

repeat

2. Input $x_p(t)$ to $W^{(o)}$ and update the network to $W^{(a)}$;

3. Input X_{val} to $W^{(a)}$, compute $L_{val}^{(a)}(t)$;

4.

for $d = 0$; $d < dim$; $d + 1$ **do**

Input $x_p(t)[d] + \Delta x_p(t)[d]$ to $W^{(a)}$

Input $x_p(t)[d] - \Delta x_p(t)[d]$ to $W^{(a)}$

Compute $L_{val}^{(a)}(t)[d]$;

end for

5. $x_p(t+1) = x_p(t) + \alpha \cdot \text{sign}(\nabla_{x_p} L_{val}^{(a)}(t))$;

6. $t = t + 1$;

until $|L_{val}^{(a)}(t) - L_{val}^{(a)}(t-1)| < L_{th}$

Output: Poisoned sample $x_p(t)$ with label y_p .

Algorithm 2 Generative method – FastGPA.

Input: Training set $X_{train} = \{x_i \mid i \in [1, n]\} \subset \mathbb{R}^{dim}$ (dim is the dimension of the data, n is the total number of the training samples), validation set for generating poisoned sample $X_{val} = \{x_i \mid i \in [1, valSize]\}$, validation set size $valSize$, attacking class y_p , generating rate α , loss factor ϵ , threshold of loss difference L_{th} .

Do:

1. Initialize and pre-train $W^{(o)}$ with X_{train} . Initialize $\{x_p(0), y_p\}$ pair. Initialize and pre-train $G(0)$ with $x_p(0)$. Set iteration $t = 0$ and $L_{val}^{(a)}(0) = 0$.

repeat

2. Input $x_p(0)$ to $G(t)$ and get $x_p(t)$

3. Input $x_p(t)$ to $W^{(o)}$ and update the network to $W^{(a)}$;

4. Input X_{val} to $W^{(a)}$ and compute $L_{val}^{(a)}(t)$ and $\{\nabla_{\hat{x}_i} L_{val}^{(a)}(t)\}_{i=1}^{valSize}$;

5. Update $G(t)$ with weighted gradient in Equation 11.

6. $t = t + 1$;

until $|L_{val}^{(a)}(t) - L_{val}^{(a)}(t-1)| < L_{th}$

Output: Poisoned sample $x_p(t)$ with label y_p .

Figure 1. (Left) Numerical gradient method algorithm; (Right) Generative method – FastGPA algorithm

Generative method – FastGPA. The poisoned sample generated by the numerical gradient method can effectively degrade the accuracy of the target model. However, the associated sample generation process is time-consuming due to the element-wise derivative computation. Moreover, this computational cost is proportional to the dimension of the input sample as well as the complexity of the target model. Hence, the scalability of the numerical gradient method is a severe issue, especially considering the rapid growth of the sizes of the neural network models and their training sets in real applications. To solve this issue, we propose a generative method for poisoned sample generation, namely, FastGPA, which can avoid the costly gradient computation and therefore substantially increase the poisoned sample generation rate.

The key idea of FastGPA is to iteratively train an additional model, i.e., the generator, or G , to generate x_p . The final G is used as the designated generator for the evaluation. In this work, a general autoencoder is chosen as G . The procedures of FastGPA are summarized in Figure 1 (Right).

Figure 2 illustrates the iteration t and $t + 1$ of FastGPA. We assume that the prerequisites of numerical gradient method and FastGPA are almost the same to an attacker, except for a pre-trained autoencoder serving as G in FastGPA. We compute $\nabla_{\Sigma \hat{x}_i} L_{val}^{(a)}(t)$ and obtain the weighted gradient to update G . In this way, the gradients w.r.t. the poisoned sample do not have to be computed element-wisely as they do in Algorithm 1. $W^{(o)}$ thus only updates once per iteration (from $W^{(o)}$ to $W^{(a)}(t)$). As a result, all gradients are first derivatives, avoiding the computation of the second partial derivative. The time consumption of poisoned sample generation is greatly reduced, especially when $W^{(o)}$ is complex. FastGPA terminates when two consecutive losses meet the criterion (e.g., smaller than a preset threshold). The threshold and the generation rate should be carefully selected to avoid local minimum.

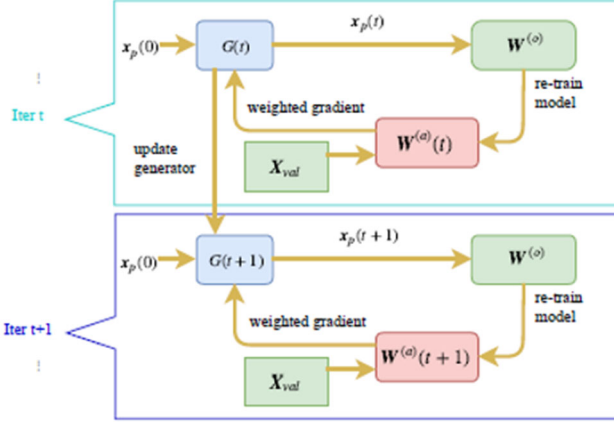


Figure 2. An overview of FastGPA

One critical problem is how to design the gradient to update $G(t)$. Here, we propose to pass a weighted summed gradients of $W^{(a)}$ to G :

$$\nabla_{x_p(t)} L_{gen}(X_{val}, x_p(t)) = \epsilon \cdot \sum \beta_i \nabla_{\hat{x}_i} L_{val}^{(a)}(t) \quad (10)$$

This gradient considers all the samples in the validation set, thus the generator will be updated such that retraining on x_p will increase the losses of these samples. The coefficient β_i is used to control the weights of these validation samples. To enhance the generalization of the poisoned sample, we choose to use the similarity of this sample between the original x_p . Similar samples will have smaller β_i . Specifically, we use L1 norm as $\beta_i = \|x_p(t) - \hat{x}_i\|_1$. In such a way, the generated poisoned sample can influence more on far away samples, meaning that this sample is more likely to compromise the model.

Furthermore, we propose to use the difference of the two losses from two consecutive iterations to build a reward function and combine with the above gradients as:

$$\nabla_{x_p(t)} L_{gen}(X_{val}, x_p(t)) = \epsilon \cdot \text{sign}(L_{val}^{(a)}(t-1) - L_{val}^{(a)}(t)) \sum \beta_i \nabla_{\hat{x}_i} L_{val}^{(a)}(t) \quad (11)$$

The idea of reward function comes from the reinforcement learning. It judges whether the previous update is positive or not, and the gradients of $W^{(a)}$ determine the directions of the gradient of G . If the previous update is negative, the current one will be punished by the newly computed reward. This helps stabilize the training of the generator which can thus converge better.

3.1.2 Activation Attack.

One of the more difficult threat models for an adversary operating on CNNs is a blackbox targeted evasion attack. In this scenario, the adversary only has access to inputs and outputs, with no working knowledge of the underlying weights or architecture of the target model. Even more, the adversary may choose the target class the oracle network misclassifies to. In this work, we design a blackbox targeted adversarial attack for deep CNN models. We design the attack using the property of adversarial transferability, in which examples that are adversarial to one model will often be adversarial to other models. The unique aspect of this attack is that it explicitly perturbs the feature space of a deep model with the intentions of creating more transferable adversarial examples. The intuition comes from the observation that intermediate features of well-trained models are transferable [4]. Thus, perturbations of intermediate features may also be transferable.

Methodology. To test this hypothesis, we design the Activation Attack (AA), as shown in Figure 3. By perturbing the source image, the algorithm drives the layer L activations of a whitebox model on the source image, towards the layer L activations of that model for a target image. The net effect is perturbations in feature space. We test the transferability of the perturbed features by feeding the perturbed image into the blackbox model. These perturbations then have a not-directly-observable effect on the layer of the blackbox model that has learned those same features.

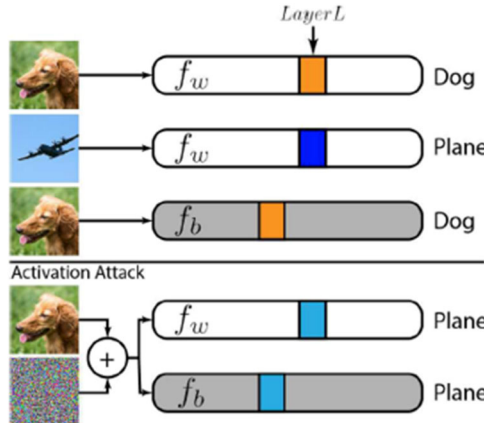


Figure 3. Illustration of Activation Attack. Given that the whitebox model (f_w) and blackbox model (f_b) are initially correct, the attack drives the layer L activations of the dog image towards the layer L activations of the plane. After the attack, the dog’s activations are similar to the plane’s and the perturbed image is classified as a plane to f_w and f_b .

Transferability metrics. In this work we will define success through four metrics: *error rate* (error), *untargeted transfer rate* (uTR), *targeted success rate* (tSuc), *targeted transfer rate* (tTR). Importantly, we assume that all the examples are correctly classified by the whitebox and blackbox model, and an attack strength of $\epsilon = 0$ means no attack. We will denote the whitebox model as the function f_w and the blackbox model as f_b , both of which output a classification prediction. We define our original dataset $D_{orig} =$

$\{(x^{(1)}, y_{true}^{(1)}), \dots, (x^{(N)}, y_{true}^{(N)})\}$ as a set of N data/label pairs for which $f_b(x^{(i)}) = f_w(x^{(i)}) = y_{true}^{(i)}$. For each attack, we create an adversarial dataset $D_{adv} = \{(x_{adv}^{(1)}, y_{target}^{(1)}, y_{true}^{(1)}), \dots, (x_{adv}^{(N)}, y_{target}^{(N)}, y_{true}^{(N)})\}$ where each data in D_{orig} has been perturbed by a targeted attack method on f_w , making it an adversarial example.

The error rate (error) or fooling rate of an attack is the percentage of adversarial examples generated with f_w that are misclassified by f_b . In other words, error is the percentage of examples in D_{adv} for which $f_b(x_{adv}) \neq y_{true}$. Larger error indicates more effective attacks.

The *untargeted transfer rate* (uTR) is the rate at which the examples that fool the whitebox model also fool the blackbox model. Note, fooling in this context means the prediction does not equal the true label. To measure uTR, we define $D_{uTR} \subseteq D_{adv}$ which contains only the elements of D_{adv} that are misclassified by f_w . Thus,

$$uTR = \frac{1}{|D_{uTR}|} \sum_{(x_{adv}, y_{true}) \in D_{uTR}} \mathbb{1}[(f_b(x_{adv})) \neq y_{true}], \quad (12)$$

where $\mathbb{1}$ is the indicator function that is 1 if the condition is true and 0 otherwise. This metric intuitively encodes the likelihood that a successful untargeted adversarial example to the attacker's whitebox model will also be adversarial to the blackbox model.

Since the attacks considered are all targeted, we measure the *targeted success rate* (tSuc). tSuc is the rate at which adversarial examples generated with f_w are classified by f_b as the target label. In other words, tSuc is the percentage of examples in D_{adv} for which $f_b(x_{adv}) = y_{target}$. The larger tSuc, the more effective the attack at generating targeted examples for the blackbox model.

The final metric is *targeted transfer rate* (tTR), which measures the rate at which successful targeted adversarial examples measured on the whitebox model are also successful targeted examples on the blackbox model. For this, we define $D_{tTR} \subseteq D_{adv}$ (also a subset of D_{uTR}) which contains all elements of D_{adv} that are misclassified by f_w as the specified target label. Formally,

$$tTR = \frac{1}{|D_{tTR}|} \sum_{(x_{adv}, y_{target}) \in D_{tTR}} \mathbb{1}[(f_b(x_{adv})) = y_{target}]. \quad (13)$$

tTR encodes a likelihood that a successful targeted example observed on the whitebox model will be a successful targeted example to the blackbox model. Current targeted attack literature commonly measures error and tSuc. We introduce uTR and tTR as new metrics for attacks that are useful if the attacker wants to maximize his chance of success in a limited number of attempts. Also note, even though the attacks are targeted, we also measure untargeted statistics as they are still relevant to the power of the attacks.

Loss function. The AA loss function is defined as the Euclidean distance between the vectorized source image activations and vectorized target image activations at some layer L . Let f_L be a truncated version of f_w (the whitebox model) that takes an image as input and outputs the activations at layer L . So $A_s^L = f_L(I_s)$ are the source image (I_s) activations and $A_t^L = f_L(I_t)$ are the target image (I_t) activations at L . The loss function J_{AA} between two images is then,

$$J_{AA}(I_t, I_s) = \|f_L(I_t) - f_L(I_s)\|_2 = \|A_t^L - A_s^L\|_2 \quad (14)$$

The intuition behind the AA loss function is to make the source image closer to an image of the target class in feature space. The implications/assumptions of this loss are three-fold. First, *adjustments of deep feature space representations have a sizable impact on the classification result*. Since we are not explicitly optimizing on classification loss, we rely on a byproduct of feature space perturbations being significant classification disruption. Because the feature space representation is so large and uninterpretable, it is not immediately obvious that this will be the case. Also, due to the size (i.e. number of parameters) of the feature space, we must assume that constrained image domain perturbations will be able to move the original sample to be close enough to the target sample (in feature space), as to be within regions of the target class.

The second major assumption made with this loss function is: *since intermediate layer features of deep models have shown to be transferable, explicit attacks in feature space will yield transferable adversarial examples*. Because modern deep models are unintelligible, there is no way of measuring exactly what features are captured and learned in each layer of a model. Thus, there is no way of definitively knowing if two models have learned similar feature sets, especially in deep and highly abstract layers. This attack principally assumes that deep layers of different deep models have learned similar features, and thus perturbing these highly abstract features in one model will perturb the same features in the other model. This is a reasonable assumption because in transferability attacks, we train the whitebox model on data from the same distribution as the blackbox model's training data. Thus, we expect that the models have learned similar hierarchical feature sets in order to properly model the classes of data. Since each model architecture differs in complexity, number of layers, and overall architecture, finding the layers at which features are most transferable is left to experimentation.

The third major assumption is regarding the learned decision boundaries, and class orientations in feature space. Specifically, we assume that *two different models trained with data from the same distribution learn similar decision boundaries and class orientations*. This is particularly relevant for the targeted attacks because for a transferred targeted example to be successful, the regions of that target class in feature space must have the same orientation w.r.t. the source image. In other words, if we start with the source example's feature space representation in the whitebox model and move it in the direction of the target sample, this assumption states that the direction of movement is the same (or at least similar) in the blackbox model.

Attack algorithm. The perturbation mechanism is like that of the L_∞ constrained iterative gradient sign attack with momentum (TMIFGSM) [5]. Our attack algorithm iteratively perturbs the source image using the sign of the momentum term, where momentum is calculated as the weighted accumulation of gradients. The difference here is that gradients are not computed against classification loss. Rather, they are computed against Equation 14. Also, gradients flow backward starting from layer L . Thus, momentum is calculated as

$$m_{k+1} = m_k + \frac{\nabla_{I_k} J_{AA}(I_t, I_k)}{\|\nabla_{I_k} J_{AA}(I_t, I_k)\|_1} \quad (15)$$

where $m_0 = \mathbf{0}$ and I_k is the perturbed source image at iteration k . Note, $I_0 = I_s$. The perturbation method for this targeted L_∞ constrained Activation Attack is

$$I_{k+1} = \text{Clip}(I_k - \alpha * \text{sign}(m_{k+1}), 0, 1). \quad (16)$$

Notice, the perturbed image is always clipped to range $[0, 1]$, as to maintain the distribution of the original image. The intrinsic meaning of Equation 16 is that we are slightly adjusting each pixel of the image in the direction that will minimize our J_{AA} loss. The momentum term is used to intuitively denoise or smooth the gradient directions. Also, keep in mind, we are perturbing the image with the explicit intentions of altering the feature space representation at some layer L . Thus, any effects on classification are implicit, as we are not specifically accounting for classification loss.

There are also some hyperparameters to set for the algorithm. Since this is an iterative algorithm, we must choose the number of iterations K to perturb for, the total perturbation amount ϵ , and the per-iteration perturbation amount α . In all tests, we set $K = 10$, vary the ϵ , and set $\alpha = \epsilon/K$.

3.1.3 Feature Distribution Attack.

One perspective on adversarial attacks is that adversarial noise is a direction in which to “move” the natural data. In standard attacks which directly use the classification output, the noise points in the direction of the nearest decision boundary at the classification layer. In this work, our crafted noise points in a direction that makes the data “look like” a sample of another class in intermediate feature space. Intuitively, if we can alter the representation in a layer whose features are representative of the data for the given task, but not specific to the model, the adversarial example may transfer better (to unobserved architectures) than attacks derived from logit-layer information.

Figure 4 (top) illustrates the feature distribution modeling of a DNN, which is the core mechanism of the attack. f is a pre-trained substitute whitebox model to which we have full access. The true target blackbox model is not shown, but we only assume limited query access and that it has been trained on ImageNet-1k [6]. Adversarial examples are then generated on the whitebox model and transferred to the blackbox model. The novelty of the attack comes from the explicit use of class-wise and layer-wise feature distributions. In Figure 4 (top), an auxiliary Neural Network (NN) $g_{l,c}$ learns

$p(y = c|f_l(x))$, which is the probability that the layer l features of the whitebox model, extracted from input image x , belong to class c . The attack uses these learned distributions to generate targeted (or untargeted) adversarial examples by maximizing (or minimizing) the probability that the adversarial example is from a particular class’s feature distribution (Figure 4 (bottom)). We also use these learned distributions to analyze layer-wise and model-wise transfer properties, and to monitor how perturbations of the input change feature space representations. Thus, we gain insights on how feature distributions evolve with layer depth and architecture.

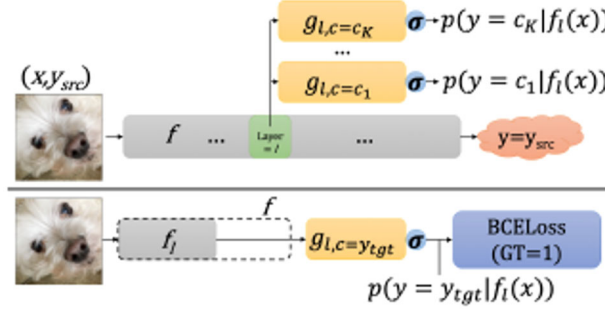


Figure 4. (Top) Given a pre-trained whitebox model f , we capture the layer-wise and class-wise feature distributions with binary neural networks $g_{l,c}$, aiming to model the probability that the layer l features extracted from input x are from the class c feature distribution (i.e. $p(y = c|f_l(x))$); (Bottom) Forward pass for FDA targeted attack

Methodology. We assume to have a set of training data and a pre-trained model f from the same task as the target blackbox model (i.e., the ImageNet-1k training set and a pre-trained ImageNet model). To model the feature distributions for f , we identify a set of classes $\mathcal{C} = \{c_1, \dots, c_K\}$ and a set of layers $\mathcal{L} = \{l_1, \dots, l_N\}$ that we are keen to probe. For each layer in $\mathcal{L}$, we train a small, binary, one-versus-all classifier g for each of the classes in $\mathcal{C}$, as shown in Figure 4 (top). Each binary classifier is given a unique set of parameters and referred to as an auxiliary model $g_{l,c}$. The output of an auxiliary model represents the probability that the input feature map is from a specific class $c \in \mathcal{C}$. Thus, we say that $g_{l,c}(f_l(x))$ outputs $p(y = c|f_l(x))$, where $f_l(x)$ is the layer l feature map of the pre-trained model f given input image x .

Once trained, we may leverage the learned feature distributions to create both targeted and untargeted adversarial examples. Here, we focus mostly on targeted attacks, which are considered a harder problem (especially in the blackbox transfer case when we do not have access to the target model’s gradients). Recall, the goal of a targeted attack is to generate an adversarial noise that when added to a clean sample x of class y_{src} , the classification result of $x + \delta$ is a chosen class y_{tgt} . The key intuition for our targeted methods is that if a sample has features consistent with the feature distribution of class c at some layer of intermediate feature space, then it will likely be classified as class c . Although not shown in the objective functions for simplicity, for all attacks the

adversarial noise is constrained by an ℓ_p norm (i.e., $\|\delta\|_p \leq \epsilon$), and the choice of layer l and target class label y_{tgt} are chosen prior to optimization.

We propose three targeted attack variants. The most straightforward variant, called *FDA*, finds a perturbation of the “clean” input image x that maximizes the probability that the layer l features are from the target class distribution:

$$\max_{\delta} p(y = y_{tgt} | f_l(x + \delta)) \quad (17)$$

We stress that unlike standard attacks that use output layer information to directly cross decision boundaries of the whitebox, our *FDA* objective leverages *intermediate feature distributions* which do not implicitly describe these exact boundaries.

In addition to maximizing the probability that the layer l features are from the target class distribution, the *FDA+ms* variant also considers minimizing the probability that the layer l features are from the source class y_{src} distribution (*ms* = minimize source):

$$\max_{\delta} \lambda p(y = y_{tgt} | f_l(x + \delta)) - (1 - \lambda) p(y = y_{src} | f_l(x + \delta)) \quad (18)$$

Here, $\lambda \in (0,1)$ weights the contribution of both terms and is a fixed positive value.

Similarly, the *FDA+fd* variant maximizes the probability that the layer l features are from the target class distribution while also maximizing the distance of the perturbed features from the original features (*fd* = feature-disruption):

$$\max_{\delta} p(y = y_{tgt} | f_l(x + \delta)) + \eta \frac{\|f_l(x + \delta) - f_l(x)\|_2}{\|f_l(x)\|_2} \quad (19)$$

In other words, the feature-disruption term, with a fixed $\eta \in \mathbb{R}_+$, prioritizes making the layer l features of the perturbed sample maximally different from the original sample.

The additional terms in *FDA+ms* and *FDA+fd* encourage the adversarial sample to move far away from the starting point, which may intuitively help in generating (targeted) adversarial examples. Also, notice that *FDA+ms* requires the modeling of both the source and target class distributions, whereas the others only require the modeling of the target class distribution.

The trained auxiliary models afford a way to construct a fully differentiable path for gradient-based optimization of the objective functions. Specifically, to compute *FDA* adversarial noise from layer l , we first build a composite model using the truncated whitebox model f_l and the corresponding layer’s auxiliary model $g_{l,c=y_{tgt}}$ for the target class y_{tgt} , as shown in Figure 4 (bottom). The loss is calculated as the Binary Cross

Entropy (BCELoss) between the predicted $p(y = y_{tgt} | f_l(x))$ and 1. Thus, we perturb the input image in the direction that will minimize the loss, in turn maximizing $p(y = y_{tgt} | f_l(x))$. For optimization, we employ iterative gradient descent with momentum.

3.1.4 Multi-Layer Feature Distribution Attack.

Our recent innovation from **Section 3.1.3** (FDA) shows the benefits of crafting blackbox transfer attacks based on intermediate layer representations, rather than simply optimizing the classification loss at the output layer. In this work, we propose to significantly improve the FDA method by extending it into a more flexible framework to allow for perturbations across the intermediate feature space, including the output layer. Our method relies on modeling the layer-wise and class-wise feature distributions of the whitebox model via auxiliary networks. We then optimize adversarial noise using the auxiliary models from across the deep feature space. The critical observation in this work is that by enforcing that a perturbed sample “looks-like” a target class sample in multiple layers across the depth of the whitebox model, the resulting sample is likely to have a much higher transfer rate than if we only considered a single intermediate layer or the output layer in isolation.

Methodology. For an intuition for how the attack works, consider Figure 5, which shows the forward pass required to generate adversarial noise given a pre-trained DNN f . Each $g_{l,y_{tgt}}$ is a feature distribution model that estimates the probability that the layer l feature map $f_l(x)$ is from a sample of class y_{tgt} , i.e., $p(y_{tgt} | f_l(x))$. For a chosen y_{tgt} and set of layers, we accumulate the losses w.r.t. y_{tgt} at each intermediate layer and the output layer. By optimizing the sum, we are noising x with δ such that $x + \delta$ lies in high probability regions of the target class at several layers across feature space. We find that this method significantly improves transferability of the generated adversarial samples.

Our proposed attack builds on FDA+*fd*, the top performing variant from **Section 3.1.3**, which we will now refer to as FDA. A key contribution of this work is the extension of FDA to include multi-layer information. One way to do so is to incorporate the whitebox model’s output prediction as part of the attack objective. Let $H(f(x), y)$ be the standard cross-entropy loss between the predicted probability distribution $f(x)$ and the target distribution y (commonly a one-hot distribution). We include this term in the FDA objective to create FDA+*xent*, as described by

$$\max_{\delta} p(y = y_{tgt} | f_{\ell}(x + \delta)) + \eta \frac{\|f_{\ell}(x + \delta) - f_{\ell}(x)\|_2}{\|f_{\ell}(x)\|_2} - \gamma H(f(x + \delta), y_{tgt}), \quad (20)$$

Where $\gamma > 0$, weighting the contribution of the cross-entropy term. This attack objective optimizes the noise such that the layer l feature map is in a high-probability region of the target class, and that the output prediction of the whitebox model is of the target class. To further generalize the multi-layer framework, we extend the objective function to allow for optimization in multiple intermediate layers. We call this new attack FDA(N):

$$\max_{\delta} \sum_{\ell \in \mathcal{L}} \lambda_{\ell} \left[p(y = y_{tgt} | f_{\ell}(x + \delta)) + \eta \frac{\|f_{\ell}(x + \delta) - f_{\ell}(x)\|_2}{\|f_{\ell}(x)\|_2} \right], \quad (21)$$

where $\sum_{\ell \in \mathcal{L}} \lambda_{\ell} = 1, \lambda_{\ell} > 0$, and the N in the name refers to the number of layers in L. Both the *+xent* and multi-intermediate-layer extensions significantly increase the capability and flexibility of the method, while maintaining the intuition of attacking primarily based on feature space information. These two extensions can also be straightforwardly composed to form an FDA(N)+*xent* attack.

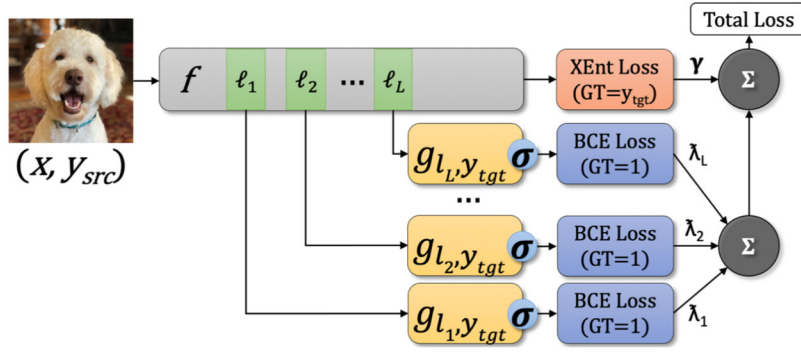


Figure 5. Visualization of forward pass construction to optimize our feature space attack objective

The final step in the methodology is the optimization of the above objective functions. Practically, we leverage the autograd feature of PyTorch by assembling the forward pass of the attack as shown in Figure 5. This step involves connecting the required auxiliary models to the corresponding layers of f and computing the final loss with proper weighting. We then use an iterative projected gradient descent with momentum procedure to apply the adversarial noise to the input while maintaining the norm and natural image constraints.

3.1.5 No Label Space Overlap Attack.

The intrigue of blackbox adversarial attacks is that only limited knowledge of the target model is assumed, making them more difficult but also more practical. Of the several blackbox techniques, transfer-based attacks have become popular because they do not rely on repetitively querying the target model during attack generation [5]. However, a glaring weakness in the design and evaluation of current transfer attacks is the implicit assumption that *the attacker’s source model is trained on the exact same dataset as the target blackbox model!* This oversight is largely due to the common practice of testing transfers exclusively between pre-trained models from the same benchmark dataset (e.g., MNIST, CIFAR-10, and ImageNet) and has likely led to overly optimistic estimates of transferability that may not translate to real-world settings, in which an attacker may not have access to the target model’s data.

In this work, we explore the potency of transfer attacks in a much more restrictive setting, where this shared dataset assumption is eliminated. Specifically, we investigate the feasibility of targeted transfer attacks when not only is there zero training data overlap, but also no label space overlap between the source and target model training distributions. Upon initial consideration, the premise of this disjoint label space transfer scenario may seem absurd. The formulation of most contemporary targeted attacks boils down to maximizing the likelihood of a target class. However, if the attacker’s whitebox model does not have the target class in its label space, how can targeted transfer be optimized for? Our primary goal is thus to probe whether it is possible to achieve attack success in such settings, and if so, to what extent.

Methodology. A basic overview of our attack setup and procedure is shown in Figure 6. Given a whitebox and blackbox model that are trained on unique class sets, we propose first constructing a correspondence matrix from the blackbox’s predictions on samples from the whitebox distribution, to identify any relationships that exist between classes across label spaces. This is followed by an adversarial noise generation phase, which leverages highly related proxy classes from the whitebox’s label space to induce the desired targeted response by the blackbox. Intuitively, our proposed attack methodology seeks to exploit a shared feature set, along with a set of potentially shared adversarial sub-spaces [7] between the two training distributions, whose presence is suggested by high responses in the correspondence matrix.

The first phase of our attack methodology is to model the relationships between the classes of the whitebox and blackbox. Note, this step is unique to our disjoint label space setting, as in previous works the class relationships are explicitly encoded by the intersection of the label sets. We accomplish this task by forward passing a limited amount of data from each category of the whitebox model’s training distribution through the target model and recording the predictions in a class correspondence matrix. Intuitively, this matrix describes how the blackbox model responds to the presence of whitebox data features, which will become relevant when constructing adversarial perturbations. Note, we consider this step to occur “offline” w.r.t. the actual attack, as it is only necessary to do once (before computing any adversarial examples) and does not involve inputting adversarially perturbed data through the target model.

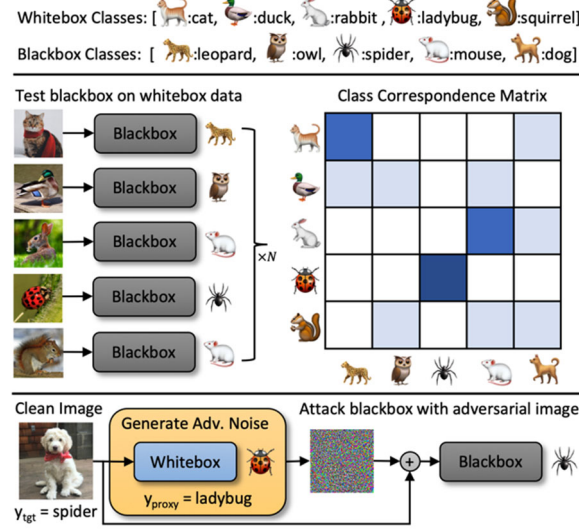


Figure 6. Overview of attack setup and procedure

The second phase of our methodology is to construct the targeted adversarial perturbations. We start with a clean image x and target class y_{tgt} from the blackbox distribution. The goal is then to compute adversarial noise δ using the whitebox model such that the blackbox model classifies the adversarial example $x + \delta$ as y_{tgt} . To do this, we first index the class correspondence matrix with y_{tgt} to find a highly correlated proxy class y_{proxy} from the whitebox distribution that can be used to represent y_{tgt} during attack generation. Intuitively, since y_{tgt} and y_{proxy} images are interpreted similarly by the blackbox model, imparting the features of y_{proxy} onto x through an adversarial noising process may in-turn cause the blackbox model to misclassify the adversarial example as y_{tgt} . We explore two fundamentally different approaches for optimizing the adversarial noise: decision-space methods and feature-space methods.

Decision-space Attack. Decision-space methods work to directly manipulate the output-layer signals of the whitebox model. Often, this is accomplished by minimizing the classification loss w.r.t. a designated target class (in our case y_{proxy}). Thus, the adversarial perturbation is quasi-optimal for the whitebox, as it is designed to traverse exact decision boundaries. Let f represent the whitebox model and $f(x)$ be the predicted probability distribution over its set of classes. We use the powerful Targeted Momentum Iterative Method (TMIM) [5] as a representative of decision-space attacks, whose optimization objective is

$$\min_{\delta \in \mathcal{S}(x; \epsilon)} H(f(x + \delta), y_{proxy}). \quad (22)$$

Here, $H(f(x + \delta), y_{proxy})$ is the cross-entropy between the whitebox's predicted distribution and y_{proxy} . The constraint $\mathcal{S}(x; \epsilon)$ defines an allowable perturbation set w.r.t. x , often to keep δ imperceptible. Optimizing this objective results in adversarial

examples capable of fooling the whitebox model into predicting y_{proxy} with high confidence. We hypothesize that by pushing $x + \delta$ into a high probability region of y_{proxy} in the whitebox, these decision-based attacks may in-turn cause the blackbox to regard $x + \delta$ as y_{tgt} .

Feature-space Attack. As the name suggests, feature space methods compute adversarial noise using the intermediate feature information of the whitebox. Rather than optimizing to explicitly cross decision boundaries, these methods make the adversarial examples “look like” the target class (or in our case y_{proxy}) in feature space. In this work, we use the multi-layer Feature Distribution Attack (*FDA*) (see **Section 3.1.4**) as our representative feature space method. For setup, we first train the necessary auxiliary feature distribution models for each proxy class at a specified set of whitebox model layers $\mathcal{L} = \{\ell_1, \dots, \ell_N\}$. Note, the class c , layer ℓ auxiliary model inputs the whitebox’s layer ℓ feature map $f_\ell(x)$, and outputs the probability that it is from an input of class c (i.e., it outputs $p(y = c | f_\ell(x))$).

Using the trained auxiliary models, the *FDA* attack objective function can then be defined as

$$\max_{\delta \in S(x; \epsilon)} L_{FDA}(f, x, y_{\text{proxy}}, \delta, \mathcal{L}, \eta), \quad (23)$$

where

$$\begin{aligned} L_{FDA}(f, x, y, \delta, \mathcal{L}, \eta) \\ = \frac{1}{|\mathcal{L}|} \sum_{\ell \in \mathcal{L}} p(y | f_\ell(x + \delta)) + \eta \frac{\|f_\ell(x + \delta) - f_\ell(x)\|_2}{\|f_\ell(x)\|_2}. \end{aligned} \quad (24)$$

By maximizing L_{FDA} , the adversarial noise: (1) maximizes the likelihood that intermediate features from across the whitebox’s feature hierarchy belong to the proxy class; and (2) enforces that the perturbed image’s feature map is significantly different from the clean image’s feature map, as measured by a normalized L_2 distance.

Our intuition for why this method has promise stems from the discussed potential overlap of features in the whitebox and blackbox data distributions. Particularly, if there exist strong (target, proxy) relationships in the class correspondence matrix, we posit that this is clear evidence that the blackbox model has learned features that appear in the whitebox data distribution. Thus, the *FDA* attack is well-phrased to manipulate any shared features that may exist between a given (target, proxy) class pair. Further, we hypothesize that since the perturbation objective of *FDA* is detached from the exact decision boundary structure of the whitebox (which is irrelevant to the blackbox model anyway), it may yield higher transferability because the optimization is focused on the feature compositions.

3.1.6 Snooping Attacks on Deep Reinforcement Learning.

Most of the recent work in the adversarial machine learning space deals with the domain of image classification. As of now, adversarial attacks against Deep Reinforcement Learning (DRL) agents primarily consider the white-box threat model, in which the adversary is assumed to have access to the target model's architecture and parameters. To our knowledge, the study of blackbox attacks against DRL has been limited to the acknowledgement of the transferability of adversarial examples across policies and algorithms. Thus, the current method for blackbox attacks against DRL agents involves training a surrogate agent, which makes the rather unrealistic assumption that the adversary has unbounded access to the environment. In general, the blackbox threat model in Reinforcement Learning (RL) is more complex than it is in supervised learning. First, in the data generating process for supervised learning the model draws i.i.d. samples from a static dataset. In RL, data generation is in the form of an environment from which new data samples are conditioned on the current environment state and the agent's actions. This means that an RL adversary cannot trivially approximate the training data generating process the way a supervised learning adversary can. Also, the RL problem consists of an exchange of three signals between the environment and agent (state, action, reward). Even if an RL adversary seeks to approximate the agent or environment using some other method (e.g., inverse RL), they would need access to all signals.

In this work, we investigate threat models for DRL under the assumption that the adversary does not have access to the environment that the target agent interacts with. Specifically, we explore the potential effectiveness of an adversary that only can eavesdrop on a subset of the RL signals at each time step. We dub these the snooping threat models. For a real-world example of the snooping threat models of a vision-based DRL system, consider a camera-equipped mobile system such as an autonomous vehicle. These systems typically send the camera feed via a wireless network to a server running an instance of the trained DRL agent, which processes the images and decides how to act in the given state. As with all current adversarial threat models, the adversary is assumed to have the ability to alter the input of the model. Action signals are transmitted from the server back to the drone via a separate channel. Finally, reward signals may come from a variety of sources. Given this scenario, it is entirely possible that the adversary cannot access or eavesdrop on all signals due to private channels, encryption, etc.

Methodology. Our general attack methodology is to leverage the transferability of adversarial examples. While this concept is not fully understood, it is widely believed that transferability is primarily enabled by decision boundary similarity between models. We argue that this condition, however, generally overestimates the resources necessary to craft effective transferable adversarial examples. The snooping threat models for RL provides one such counterexample, in which the adversary cannot train a surrogate DRL model that closely resembles the target agent. Instead, we posit that if the adversary can train a proxy model that learns a task that is related to the target agent's policy to maximize reward, adversarial examples crafted to fool the proxy will also fool the agent. This implies that even proxy models with radically different decision surface

dimensionality than the target can reliably produce transferable attacks. We hypothesize that by learning a related task, the proxy will learn to extract and use input features in a similar fashion to the target agent, which is the only necessary condition for crafting transferable attacks. Thus, if an attacker can intercept the state signals, they could conceivably use this proxy model to generate strong attacks against the agent model. See Figure 7 for an overview of this threat model.

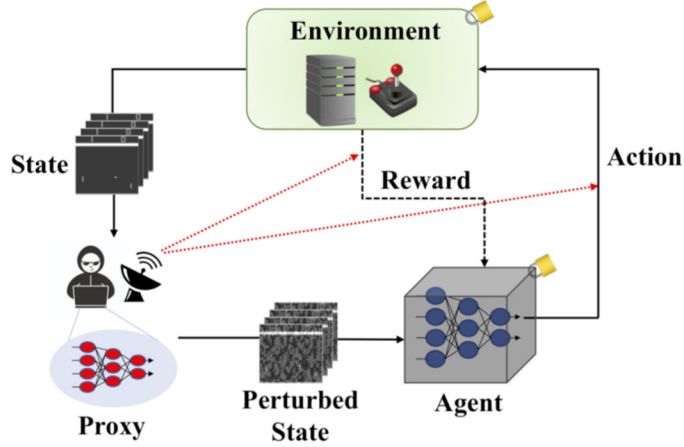


Figure 7. Snooping Threat Model Overview

We consider three different snooping threat models:

S threat model. The S threat model is the weakest snooping threat model. The adversary only has access to a stream of states and has no inclination about the motivations or capabilities of the agent. In this case, we propose a proxy that models the environment dynamics. In this setting, we can approximate the next state s_{t+1} given the current state s_t under the target agent’s policy π_T using a psychic proxy model:

$$\text{psychic}(s_t, \theta_p) \approx \mathbb{E}_{\pi_T}[P(s_{t+1} | s_t)] \quad (25)$$

Because the psychic learns the dynamics under π_T , it effectively learns an encoding of π_T in the state space. In other words, the displacement of the agent’s representation in the state space from s_t to s_{t+1} inherently encodes a representation of the action a_t taken and therefore an instance of the agent’s policy $\pi_T(s_t)$. This approximation will be noisy in general due to displacements in the state space that are not the consequence of the agent’s actions. Despite the noise, the key is that a subset of the features learned by the psychic will be similar to the features learned by the target agent.

SR threat model. In the SR threat model, we can intercept the states and snoop on the rewards. Because we do not have access to the actions, we again cannot make any presumptions about the policy π_T directly. However, by observing the rewards resulting from each state in a rollout we can get insight into the agent’s motivations by training an

assessor proxy to estimate the value V of a given state under π_T . Since the reward signal is what drives the policy π_T , the features learned to approximate V^{π_T} must correspond to the features used by the agent itself.

$$\text{assessor}(s_t, \theta_A) \approx V^{\pi_T}(s_t) \quad (26)$$

In our study we use empirical return to estimate value, meaning that we observe the immediate rewards at each time step, and when the episode is over, we retroactively compute and standardize the discounted sum of rewards at each state. During training, the assessor minimizes a Huber regression loss between predicted value and empirical value.

SA threat model. In the SA threat model, we have access to a stream of states and can snoop on the actions taken by the agent at each step. Therefore, approximating the target’s policy π_T can be done directly with supervised learning. We can train an imitator model to predict the action that the target will take at a given state as:

$$\text{assessor}(s_t, \theta_A) \approx V^{\pi_T}(s_t) \quad (27)$$

Since the imitator is a classifier, we train it with cross-entropy loss using the agent’s actions as labels. Although the imitator is trained with a different objective, it is directly related to the target agent’s goal. For value-based agents, the policy π_T is an ϵ -greedy extension of the learned Q-function and for policy-based agents, the function π_T is directly approximated. This methodology is homologous to a typical blackbox attack strategy, in which a substitute model is trained using the target model as an oracle. Note that if we were to have a continuous action space, the imitator would simply become a regression model.

We use two fast adversarial attack algorithms to generate the adversarial examples using the proxy model gradients: Fast Gradient Method (FGM) [1], and Momentum-Iterative FGM (MIFGM) [5].

3.2. Enhancing Robustness to Adversarial Attacks and Anomalous Data

3.2.1 Soteria: Provable Defense Against Privacy Leakage in Federated Learning.

Federated learning (FL) [8] is a popular distributed learning approach that enables several devices to train a shared model in a federated fashion without transferring their local data. A central server coordinates the FL process, where each participating device communicates only the model parameters on the central server while keeping local data private. Thus, FL becomes a natural choice for developing mobile deep learning applications.

Privacy preservation is the major motivation for proposing FL. However, recent works demonstrated that sharing model updates or gradients also makes FL vulnerable to inference attack, e.g., property inference attack [9] and model inversion attack [10]. Here property inference attack infers sensitive properties of training data using the model

updates, and model inversion attack reconstructs training data using model gradients. However, the essential causes of such privacy leakages have not been thoroughly investigated or explained. Some defense strategies have been presented to prevent the privacy leakage, but they incur either significant computational overheads or unignorable accuracy loss. The reason is that existing defenses are not specifically designed for the privacy leakage from the communicated local updates. The privacy issues seriously hinder the development and deployment of FL. There is an urgent necessity to unveil the essential cause of privacy leakage such that we can develop effective defenses to tackle the privacy issue of FL.

Methodology. In this work, we assume that the server in FL is malicious, and it aims to reconstruct the private training data from devices. Our key observation is the class-wise data representations of each device's data are embedded in shared local model updates, and such data representations can be inferred to perform model inversion attacks. Therefore, the information can be severely leaked through the model updates.

Based on our observation of the representation leakage from local updates, we design a defense called Soteria. Specifically, we present an algorithm to generate a perturbation added to the data representations, such that: 1) the perturbed data representations are as similar as possible to the true data representations to maintain the FL performance; and 2) the reconstructed data using the perturbed data representations are as dissimilar as possible to the raw data.

Representation leakage in FL. Our key observation is that the data representations are less entangled in the model updates. We denote a FC layer as $\mathbf{b} = \mathbf{W}\mathbf{r}$, where $\mathbf{r}$ is the input to the FC layer (i.e., the learnt data representation by previous layers), $\mathbf{W}$ is the weight matrix, and $\mathbf{b}$ is the output. Then, given a training batch $\mathbb{B}$, the gradient of the loss l with respect to $\mathbf{W}$ is:

$$\frac{1}{|\mathbb{B}|} \sum_{i=1}^{|\mathbb{B}|} \frac{\partial l^i}{\partial \mathbf{W}} = \frac{1}{|\mathbb{B}|} \sum_{i=1}^{|\mathbb{B}|} \frac{\partial l^i}{\partial \mathbf{b}^i} \frac{\partial \mathbf{b}^i}{\partial \mathbf{W}} = \frac{1}{|\mathbb{B}|} \sum_{i=1}^{|\mathbb{B}|} \frac{\partial l^i}{\partial \mathbf{b}^i} (\mathbf{r}^i)^T, \quad (28)$$

where l^i , $\mathbf{r}^i$, and $\mathbf{b}^i$ are the loss corresponding to the i -th sample, the input, and the output of the FC layer in this batch, respectively. We observe that the gradient for a particular sample is the product of a column vector $\frac{\partial l^i}{\partial \mathbf{b}^i}$ and a row vector $(\mathbf{r}^i)^T$. Suppose the training data has C labels. We can split the batch $\mathbb{B}$ into C sets, i.e., $\mathbb{B} = \{\mathbb{B}_0, \mathbb{B}_1, \dots, \mathbb{B}_C\}$, where $\mathbb{B}_k$ denotes the data samples with the k -th label. Then, the above equation can be rewritten as:

$$\frac{1}{|\mathbb{B}|} \sum_{i=1}^{|\mathbb{B}|} \frac{\partial l^i}{\partial \mathbf{W}} = \sum_{k=1}^C \left(\frac{1}{|\mathbb{B}_k|} \sum_{j \in \mathbb{B}_k} \frac{\partial l^j}{\partial \mathbf{b}^j} (\mathbf{r}^j)^T \right) \triangleq \sum_{k=1}^C \text{Grad}(\mathbb{B}_k), \quad (29)$$

where $Grad(\mathbb{B}_k)$ represents the gradient with respect to the data samples in $\mathbb{B}_k$. Figure 8 illustrates the gradient updates for a batch data in a FC layer. We observe that for data coming from different classes, the corresponding data representations tend to be embedded in different rows of gradients. If the number of classes is large in a batch, which is common in centralized training, the representations of different classes will be entangled in the gradients of this whole batch. In contrast to centralized training, the local data often covers a small number of tasks on a participating device in FL. Thus, the number of data classes C within one training batch may be very small compared to that of the centralized training. In this case, the entanglement of data representations from different classes can be significantly reduced. Such a low entanglement of data representations allows us to explicitly reconstruct the input data of each class from the gradients, because we can (almost) precisely locate the rows of data representations in the gradients.

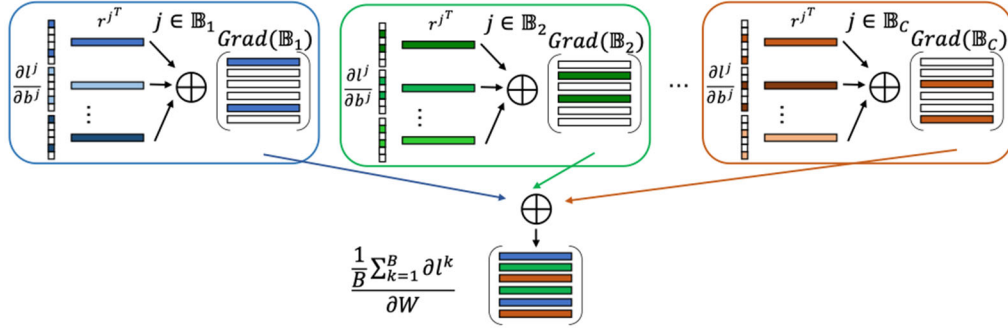


Figure 8. Illustration of the gradient updates of class-wise data in a batch

Inferring class-wise data representations. We develop an algorithm to identify the training classes and infer the data representations of each class embedded in each FC layer from the model updates. The representation inference algorithm is in a back propagation fashion. Specifically, we first identify the classes using the gradients of the last L -th layer. We denote the gradients of the L -th layer as ∇W^L . We notice that the gradient vector ∇W_i^L , which is the i -th row in ∇W^L , shows significantly larger magnitudes than gradient vectors in other rows if the data from i -th class are involved in training. Then, we can infer the data representation of the i -th class in the last layer because it linearly scales ∇W_i^L . If data representation of the i -th class in layer W^L is inferred, we can use their element values to identify the corresponding row from the $(L - 1)$ -th layer's gradients, i.e., W^{L-1} , which embeds the data representation of the i -th class in the $(L - 1)$ -th layer. In this way, we can iteratively infer data representations of the i -th class in all FC layers.

We conduct a preliminary experiment on CIFAR10 to evaluate whether our algorithm can unveil the data representation. We consider the practical non-IID settings in FL and follow the 2-class & balanced configuration [11] to construct non-IID datasets: 100 devices in total and 10 devices are randomly sampled to participate in training in each communication round. Each device holds 2 classes of data and each class has 20 samples.

We use the correlation coefficient cor between the true representation and our inferred representation to quantify the effectiveness of our proposed algorithm. The results are presented in Table 1. One can see that the correlation cor is as high as 0.99, indicating a serious representation leakage in FL. cor decreases with B or a larger number of batches in one epoch and increases as E goes lower, which validate our claim. However, cor is still higher than 0.8 in almost all cases.

Table 1. Average cor across 200 communication rounds under different settings

Local Training Configurations	FC1	FC2	FC3
E=1, B=32	0.98	0.99	0.99
E=5, B=32	0.82	0.90	0.92
E=10, B=32	0.70	0.78	0.82
E=1, B=16	0.82	0.93	0.99
E=1, B=8	0.85	0.89	0.92
E=1, B=32 (IID)	0.48	0.31	0.18

Representation leakage is the essential cause of information leak. In this section, we investigate whether the representation leakage is the essential cause of information leakage in FL. Particularly, we conduct experiments on CIFAR10 to reconstruct the input based on the existing DLG attack [10]. DLG attack requires the gradient information, and we consider three different portions of the gradients: the whole model gradients (WG), the gradients of convolutional layers only (CLG), and inferred representations using our method (Rep). As Figure 9 shows, only utilizing gradients of convolutional layers cannot successfully reconstruct the input data, but using the representation inferred by our method can reconstruct the input data as effectively as utilizing the whole gradients in terms of visual quality. This result validates that representation leakage is the essential cause of privacy leakage in FL.

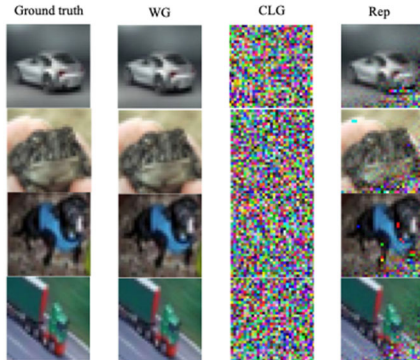


Figure 9. DLG attack results utilizing different parts of gradients

Defense from the representation perspective. Our observation shows that the privacy leakage in FL mainly comes from the representation leakage (e.g., in the FC layer). In this section, we propose a defense against such privacy leakage called Soteria. We propose to perturb the data representation in a single layer (e.g., a FC layer), which we call the *defended layer*, to satisfy the following two goals:

- 1) To reduce the privacy information leakage, the reconstructed input through the perturbed data representations and the raw input should be dissimilar.
- 2) To maintain the FL performance, the perturbed data representation and the true data representations without perturbation should be similar.

Let r and r' represent the clean data representation and perturbed data representation on the defended layer, respectively. We also define X and X' as the raw input and the reconstructed input via the perturbed data representation. To achieve Goal 1), we require that the distance between X and X' , in terms of ℓ_p norm, should be as large as possible. To reach Goal 2), we require that the distance between r and r' , in terms of ℓ_q norm, should be bounded. Formally, we have the following constrained objective function with respect to r' :

$$\begin{aligned} \text{Achieving Goal 1): } & \max_{r'} \|X - X'\|_p, \\ \text{Achieving Goal 2): } & \text{s.t. } \|r - r'\|_q \leq \epsilon, \end{aligned}$$

where ϵ is a pre-determined threshold. Note that X' depends on r' .

Prior to presenting our solution, we first define $f: X \rightarrow r$ as the feature extractor before the defended layer. We make the following assumption.

Assumption 1. *The inverse of f , i.e., f^{-1} , exists on r and r' , $\forall \|r - r'\|_q \leq \epsilon$.*

Using the inverse function theorem, we have the following lemma.

Lemma 1. *For $\forall f: X \rightarrow r$ and $f^{-1}: r \rightarrow X$, $\nabla_r f^{-1} = (\nabla_X f)^{-1}$.*

Then, our objective function can be reduced as follows:

$$\begin{aligned} r' &= \arg\max_{r'} \|X - X'\|_p, \text{ s.t. } \|r - r'\|_q \leq \epsilon, \\ &= \arg\max_{r'} \|f^{-1}(r) - f^{-1}(r')\|_p, \text{ s.t. } \|r - r'\|_q \leq \epsilon, \\ &= \arg\max_{r'} \|\nabla_r f^{-1} \cdot (r - r')\|_p, \text{ s.t. } \|r - r'\|_q \leq \epsilon, \\ &= \arg\max_{r'} \|(\nabla_X f)^{-1} \cdot (r - r')\|_p, \text{ s.t. } \|r - r'\|_q \leq \epsilon. \end{aligned} \tag{30}$$

Note that, with different choices of $\|\cdot\|_p$ and $\|\cdot\|_q$, we have different defense solutions and thus have different defense effects. In this work, we set $p = 2$, i.e., we aim to maximize the MSE between the reconstructed input and the raw input. Meanwhile, we set $q = 0$ due to two reasons: our defense has an analytical solution and is communication efficient. Specifically, our solution is to find the ϵ largest elements in the set

$\{\|r_i((\nabla_X f(r_i))^{-1})^{-1}\|_2\}$. Moreover, the learnt perturbed representation is relatively sparse and thus improves the communication efficiency. Figure 10 details the local training process with our defense on a local device.

Input: Training data $\mathbf{X} \in \mathbb{R}^{M \times N}$; Local objective function $F : \mathbb{R}^{M \times N} \rightarrow \mathbb{R}$; Feature extractor $f : \mathbf{W}_f \in \mathbb{R}^{M \times N} \rightarrow \mathbb{R}^L$ before the defended layer; The defended layer $g : \mathbf{W}_g \in \mathbb{R}^L \rightarrow \mathbb{R}^K$; Feature extractor after the defended layer $h : \mathbf{W}_h \in \mathbb{R}^K \rightarrow \mathbb{R}$; Local model parameters $\mathbf{W} = \{\mathbf{W}_f, \mathbf{W}_g, \mathbf{W}_h\}$; Learning rate η .

Output: Learnt model parameter $\mathbf{W}$ with our defense.

- 1: Initialize $\mathbf{W}$;
- 2: **for** $\mathbb{B}$ in local training batches **do**
- 3: **for** $\mathbf{X} \in \mathbb{B}$ **do**
- 4: $l \leftarrow F(\mathbf{X}; \mathbf{W})$;
- 5: $\mathbf{r} \leftarrow f(\mathbf{X}; \mathbf{W}_f)$;
- 6: $\mathbf{b} \leftarrow g(\mathbf{r}; \mathbf{W}_g)$; // e.g., $\mathbf{b} = \mathbf{W}_g \mathbf{r}$ for FC layers
- 7: $l \leftarrow h(\mathbf{b}; \mathbf{W}_h)$;
- 8: $\{\nabla \mathbf{W}_f, \nabla \mathbf{W}_g, \nabla \mathbf{W}_h\} \leftarrow \nabla_{\mathbf{W}} F(\mathbf{X}; \mathbf{W})$;
- 9: $\mathbf{r}' \leftarrow \text{Perturb_rep}(\mathbf{X}, f(\cdot; \mathbf{W}_f), \mathbf{r}, \epsilon)$;
- 10: $\nabla \mathbf{W}'_g \leftarrow \tau(l, \mathbf{b}, \mathbf{r}', \mathbf{W}_g)$; // e.g., $\nabla \mathbf{W}'_g = \frac{\partial l}{\partial \mathbf{b}} \mathbf{r}'^T$ in FC
- 11: $\nabla \mathbf{W} = \{\nabla \mathbf{W}_f, \nabla \mathbf{W}'_g, \nabla \mathbf{W}_h\}$;
- 12: $\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla \mathbf{W}$;
- 13: **end for**
- 14: **end for**

Figure 10. The local training process with our defense on a local device

3.2.2 Adversarial Outlier Exposure for Reliable Operation of SAR-ATR Models.

In the design of Automatic Target Recognition (ATR) systems for Synthetic Aperture Radar (SAR) data, we recognize that it is critical to consider an operating environment in which test samples may or may not be from one of the known classes (i.e., an “open-world” environment). In such a setting, there are two primary goals. First, the system must be capable of accurately identifying and classifying some set of In-Distribution (ID) target classes. These are categories of targets that the ATR models are trained on, and at test time, we seek generalization to their many views and perspectives. The second goal is to be capable of reliably detecting and rejecting any Out-of-Distribution (OOD) data observed during the model’s deployment. For example, if a military-vehicle classifier encounters the signature of a minivan, it should refrain from outputting a classification and instead reject the sample. In this work, we codesign a deep learning-based SAR-ATR algorithm that is both accurate on ID data and can robustly identify OOD test data under a variety of training conditions.

The basic ATR system design we consider consists of two modifiable parts. The first is a DNN model that is trained as a classifier on the training dataset, and at test time attempts to categorize the input data as one of the known classes. The second is an “OOD scoring”

mechanism that probes signals of the DNN (which is regarded as a feature extractor) and produces a decision about whether the data is ID. If the OOD prediction component determines that the data are OOD, the classifier abstains from releasing the prediction; otherwise, the prediction is released. Many related works in OOD detection consider a similar system design and focus expressly on the construction of the OOD scoring mechanic, while assuming a “good-enough” pretrained feature extractor exists. However, we posit that in a significant way, the detection is limited by the information content in the extracted features. So, in this work, we take a different approach where we focus on improving the training process of the DNN classifier/feature-extractor, for use with existing OOD scoring methods.

Intuitively, to overcome the challenge of having an implicit closed world assumption during the DNN model training, we propose to modify the learning objective to include instructions for what to do when OOD data are encountered. We use the recently released Synthetic and Measured Paired Labeled Experiment (SAMPLE) dataset [12] as the ID task. We then augment the training set with a large, diverse, and unlabeled dataset containing SAR signatures of large ships collected across a variety of sensor platforms (note, the SAMPLE dataset contains classes of landlocked military vehicles). In essence, the models are trained to be accurate on the SAMPLE data and to produce maximum entropy predictions on the ship dataset, following a concept called Outlier Exposure (OE) [13]. Importantly, we develop a novel extension of OE called Adversarial Outlier Exposure (AdvOE), which greatly improves the performance through the inclusion of active attackers within the training objective. Critically, our AdvOE method assumes no knowledge-of the expected OOD set, making its application significantly more realistic.

SAMPLE dataset. This dataset contains (measured, synthetic) *pairs* from the ten MSTAR [14] target classes. In total, there are 806 training and 539 test pairs. The measured components were taken directly from the MSTAR public release dataset while the corresponding synthetic targets were constructed via electromagnetic signature prediction from meticulous Computer Aided Design (CAD) models. The class number to class name decoding used in this work is as follows: {0:“2S1”; 1:“BMP2”; 2:“BTR70”; 3:“M1”; 4:“M2”; 5:“M35”; 6:“M548”; 7:“M60”; 8:“T72”; 9:“ZSU23”}. There are two key parameters associated with this dataset and its experiments that are referenced throughout this work. K sets the fraction of measured training data, while the test data are always 100% measured ($0 \leq K \leq 1$). For example, if $K = 0.75$ then 604/806 training pairs are represented by their measured component while the remaining 202/806 are represented by their synthetic component. As a special case, if $K = 0$ then 100% of the training samples are synthetic. To manufacture the OOD problem, parameter J is introduced to set the number of classes that are held-out from the training set ($1 \leq J \leq 8$). So, the classifiers are trained on the remaining $10 - J$ classes, and at evaluation time the test data for the held-out J classes is considered OOD while the test data for the other $10 - J$ classes is considered ID.

Methodology. Our methodological design is guided by a two-step workflow. Step 1 is to train a base classifier to be accurate on the ID dataset. Here, training is performed “offline” with no knowledge of the expected OOD testing sets. Step 2 instantiates an

OOD detector for the trained classifier, which produces a real-valued “OOD score” for each test input sample. The ID/OOD decision is then made via thresholding the score.

Training Procedure. (1) The first training procedure we consider is vanilla empirical risk minimization. Specifically, this entails minimizing a cross entropy loss over the 10 – J class ID training set D^{ID} . (2) The second training procedure we consider is a straightforward extension of empirical risk minimization called label smoothing (lblsm) [15]. This method redefines the one-hot truth label y as a smooth-label y^{LS} using equation $y_c^{LS} = y_c(1 - \alpha) + \alpha/|C|$, where α is an introduced smoothing parameter. We include lblsm at level $\alpha = 0.1$ because it has been shown to improve the generalization and calibration abilities of DNNs [15]. (3) The third training procedure is Adversarial Training (AT) [16]. We specifically use the iterative Projected Gradient Descent (PGD) variant as described in [16]. We include AT because it brings several potential benefits to our ATR system. Preliminary experiments show that it learns robust features of the data that maintain strong correlation with the true classification labels. This is particularly important for SAR tasks as the model learns representations that focus on the features of the SAR targets rather than features of the background clutter. (4) The fourth training procedure we consider is Outlier Exposure [13]. This method differs from the previous three in that we now introduce a new dataset containing only OOD samples, D_{ood}^{OE} . The OE training objective is

$$\min_{\theta} \mathbb{E}_{(x,y) \sim D_{id}} [H(f(x; \theta), y)] + \lambda \mathbb{E}_{\tilde{x} \sim D_{ood}^{OE}} [H(f(x; \theta), \mathcal{U}_C)]. \quad (31)$$

The first term in Equation 31 is the standard empirical risk minimization from training procedure (1), which works to minimize the cross-entropy loss between the predicted distribution and the one-hot label distribution for samples in D_{id} . Intuitively, this term is striving for ID accuracy. The second term encourages the DNN model to output a uniform distribution over the classes $\mathcal{U}_C$ for samples from D_{ood}^{OE} . In other words, the model is trained to have a maximum entropy output for samples not from the ID classes. Here, λ is a tune-able hyperparameter that weights the importance of the two terms. Small λ values down-weight the contribution of the outlier samples and, hence, place more emphasis on ID classification term. In contrast, large λ values prioritize the model to output uniform predictions for the OOD samples and may sacrifice the ID classification performance. In this work, we use $\lambda = 0.5$, which was shown to be a useful value for image-based classifiers while having a nondetrimental impact on accuracy.

(5) Finally, we introduce the AdvOE training procedure, which combines concepts from AT and OE. The learning objective for AdvOE is described as

$$\min_{\theta} \mathbb{E}_{(x,y) \sim D_{id}} \left[\max_{\delta \in S_{id}} H(f(x + \delta; \theta), y) \right] + \lambda \mathbb{E}_{\tilde{x} \sim D_{ood}^{OE}} \left[\max_{\delta \in S_{ood}} H(f(\tilde{x} + \delta; \theta), \mathcal{U}_C) \right]. \quad (32)$$

The critical difference between AdvOE and OE is the inclusion of the inner maximization terms to both components of the objective. As applied to the first term, the model is trained to be accurate on the ID data while learning the most robust features for

classification. As applied to the second term, the maximizer is constructing more difficult outlier samples for the DNN to learn with. By maximizing the cross-entropy loss against a uniform truth distribution, the perturbed $\tilde{x}$ samples are pushed toward lower entropy predictions, meaning they are perturbed such that they appear more ID to the DNN. The overall goal of AdvOE is to learn higher quality features of the ID data for classification while also learning a more reliable confidence estimator for OOD data via training on maximally difficult samples.

Defining D_{ood}^{OE} . From [13], the D_{ood}^{OE} set introduced in OE training is meant to be a large, diverse, unlabeled, yet somewhat realistic dataset containing samples from the wider OOD set. Given that our ID task is SAR target classification of military vehicles, we use data from the SAR-ship-dataset [17] to define a reasonably granular D_{ood}^{OE} distribution. The SAR-ship-dataset was originally proposed as a large object detection dataset that covers a variety of sensors, imaging modes, resolutions, and polarizations. In total, there are 59,535 ships from 43,819 full frame images. For our purposes, each ship was chipped from the full images using the ground truth box information. Figure 11 shows samples from both the SAMPLE and SAR-ship-dataset which make up the D_{id} and D_{ood}^{OE} sets, respectively.

OOD Detection Method. Step two of our workflow is to define the OOD detection method. In step one, we outlined five techniques for training DNNs to be accurate on the classification task. Now, we use the pretrained classifiers as “feature extractors” and define a method for producing a real-valued “OOD Score” for each test sample. In this work, we use the popular ODIN detector [18], which scores inputs based on the maximum confidence level from a temperature scaled softmax function. However, we note that our models are not limited to use with ODIN and may be combined with other OOD/OSR methods in the future. Specifically, the ODIN score for some test input x is computed as

$$S_{ODIN}(x) = \max_{i \in \mathcal{C}} \frac{\exp(g_i(x)/T)}{\sum_{j=1}^{\mathcal{C}} \exp(g_j(x)/T)}. \quad (33)$$

Here, T is a temperature hyperparameter, which is set to $T = 1000$ according to [18]. The intuition for ODIN is simple: OOD test samples should have low-confidence predictions, while ID test samples should have high-confidence predictions. The temperature scaling is a numerical trick empirically shown to help separate the ID/OOD scores. Note, $S_{ODIN}(x)$ is real valued and bounded in $[1/|\mathcal{C}|, 1]$, so to actually make an ID/OOD detection decision, a threshold β_{thresh} must be set. If $S_{ODIN}(x) \geq \beta_{thresh}$, x is considered ID and the classification prediction is given. If $S_{ODIN}(x) < \beta_{thresh}$ the sample is deemed OOD and rejected.

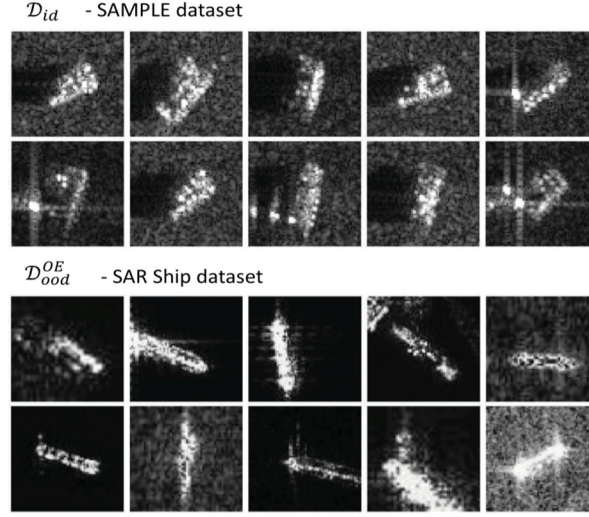


Figure 11. Samples from both D_{id} and D_{ood}^{OE} sets

Critically, there is no global optimal value for β_{thresh} , rather it must be set according to the consequences of an error (e.g., a false-negative). Therefore, to measure the performance of our detector, we leverage two metrics: Area Under the Receiver Operating Characteristic curve (AUROC) and true negative rate at a detection threshold set to achieve a 95% true positive rate (TNR@95TPR). AUROC is a threshold-independent performance metric that assess the tradeoff between true positive and false positive rates across possible thresholds. TNR@95TPR represents a situation where β_{thresh} is set to achieve 95% TPR, which constitutes the high performance at identifying ID samples, and measures what the TNR is for the OOD set at that fixed threshold.

3.2.3 Improving Out-of-Distribution Detection by Learning from the Deployment Environment.

In remote sensing (RS) applications, a core functionality is recognition, i.e., to classify a given input into one-of-several known categories. Examples of such tasks in the RS domain are scene classification in Electro-Optical (EO) satellite images and Automatic Target Recognition (ATR) in Synthetic Aperture Radar (SAR) data. Importantly, regardless of the exact data-format/modality/task such RS systems are most commonly built using Deep Neural Networks (DNNs) to perform the underlying recognition function [19]. While DNNs have shown to be exceptionally accurate in many settings [20], they have also shown to yield poor confidence calibration on anomalous/out-of-distribution (OOD) data, which leads to erroneous high-confidence predictions on input samples that do not belong to one of the known in-distribution (ID) classes [21]. This is a particularly dangerous failure-mode of a system that is expected to be deployed in an “open-world” operating environment.

Motivated by weaknesses of the DNN models and the nature of the expected operating environments, our goal in this work is to improve the safety and reliability of DNNs as

applied to open-world RS recognition tasks, such that they are accurate on ID data from the known classes while being capable of detecting and rejecting anomalous OOD data.

Methodology. The concept we use to formulate our technical approach is to let our recognition systems *quickly and efficiently learn from the operating environment*. Our methodology is centered around a proposed six step “learn-online” workflow, which executes intuitively as follows. First, a DNN classifier and associated OOD detector are trained and tuned in an offline environment, where they have no exposure to OOD samples from the deployment environment/distribution. Then, the classifier + OOD detector system is built and deployed into the operating environment where it is allowed to collect a small library of ID and OOD samples. After a label correction step, the DNN is then updated using only the collected library and is subsequently deployed back into the environment for continued operation. We remark that most related works in OOD detection focus exclusively on the offline training and tuning portion of this process and give no consideration for how the system would go about learning from the environment. Thus, a key contribution of this work is to enable OOD detector systems to learn during deployment using a limited amount of collected samples.

The classifier + OOD detector system we consider is ultimately made of two parts: (1) a DNN model that attempts to classify a given input into one of the known classes; and (2) an OOD detector which predicts each input to be either ID or OOD using some signal derived from the classifier. Note, the classifier and OOD detector are assumed to be trained/tuned in an “offline” environment, with no access to samples from the expected test OOD set. Operationally, the system functions as follows. First, a test image is input into the classifier which performs an inference step and produces a tentative prediction for the sample, attempting to classify it as one of the known classes. The OOD detector then observes the signal from the classifier and produces a real-valued ID/OOD score. If the ID/OOD score is greater than some pre-defined threshold, the input is predicted to be ID and the classifier releases its prediction. Else, the sample is deemed to be OOD, and the system abstains from releasing any prediction.

DNN training techniques. A critical detail of the classifier + OOD detector system is how the DNN classifier is trained. We now describe details of the three training methods considered in this work.

1) Standard: The first method has become a standard practice for training DNNs and is intended to be used as a baseline in this work. It performs an Empirical Risk Minimization (ERM) procedure over samples in the ID-only training distribution $\mathcal{D}_{id}$. Mathematically, this *standard* training procedure is described as

$$\min_{\theta} E_{(x,y) \sim \mathcal{D}_{id}} [H(f(x; \theta), y)]. \quad (34)$$

Here, f represents the function mapping of the DNN model, θ is the parameters, and $H(f(x; \theta), y)$ is the cross-entropy loss between the predicted distribution and the ground truth-encoded label distribution y . Intuitively, the network parameters are updated to minimize the expected cross-entropy loss (i.e., risk) for samples drawn randomly from

$\mathcal{D}_{id}$. Note, the ground truth labels y are one-hot vectors, meaning the network is encouraged to produce maximally confident predictions for each input.

2) *Vanilla Outlier Exposure*: The second training procedure we consider introduces a powerful concept called outlier exposure [13]. In addition to the ID-only empirical training distribution $\mathcal{D}_{id}$, outlier exposure introduces another empirical distribution $\mathcal{D}_{ood}^{OE}$ which contains samples from the wider OOD set. Note, it is *not* assumed that $\mathcal{D}_{ood}^{OE}$ contains samples from the expected test OOD sets. Rather, it is intended to be a large, diverse, and unlabeled set of OOD data from which we can attempt to generalize the concept of what an OOD sample is and how to handle one when encountered. The outlier exposure (OE) training objective is formulated as

$$\min_{\theta} E_{(x,y) \sim \mathcal{D}_{id}} [H(f(x; \theta), y)] + \lambda E_{\tilde{x} \sim \mathcal{D}_{ood}^{OE}} [H(f(\tilde{x}; \theta), \mathcal{U}_c)]. \quad (35)$$

Notice, the first term is the same as the *standard* training objective, which strives for high ID accuracy with high confidence predictions. The second term is the key to *OE*, as it trains the DNN to output a Uniform distribution over the classes $\mathcal{U}_c$ for OOD samples drawn from $\mathcal{D}_{ood}^{OE}$ (i.e., to have maximum-entropy outputs on OOD data). Importantly, OE has shown to significantly improve confidence calibration on OOD data in some settings.

3) *Mixup Outlier Exposure*: The third training procedure we consider is Mixup Outlier Exposure (*MixupOE*) [22]. Like *OE*, *MixupOE* is structured as a two-term objective and leverages an outlier distribution during training. However, the key difference is that instead of optimizing directly on the empirical OOD samples from $\mathcal{D}_{ood}^{OE}$, *MixupOE* leverages a *vicinal* distribution $\tilde{\mathcal{D}}_{idood}$ defined using both the $\mathcal{D}_{id}$ and $\mathcal{D}_{ood}^{OE}$ empirical distributions and a mixup operation. The basic *MixupOE* training objective is defined as

$$\min_{\theta} E_{(x,y) \sim \mathcal{D}_{id}} [H(f(x; \theta), y)] + \lambda E_{\tilde{x} \sim \mathcal{D}_{ood}^{OE}} [H(f(\tilde{x}; \theta), \mathcal{U}_c)]. \quad (36)$$

Like before, the first term strives for high ID accuracy with confident predictions. To understand the second term, we first discuss the construction of samples in $\tilde{\mathcal{D}}_{idood}$.

The mixup operation is defined as

$$\begin{aligned} \text{mixup}((x_1, y_1), (x_2, y_2), \alpha) &= (\tilde{x}, \tilde{y}) \\ &= (\lambda x_1 + (1 - \lambda)x_2, \lambda y_1 + (1 - \lambda)y_2) \text{ for } \lambda \\ &\sim \text{Beta}(\alpha, \alpha). \end{aligned} \quad (37)$$

Notice, the output virtual sample is a linear interpolation of the two input samples in both the data space and the label space; where the α input parameterizes a Beta distribution from which the linear mixing coefficient λ is sampled from. Putting it all together, the virtual samples in $\tilde{\mathcal{D}}_{idood}$ can be described as interpolations between ID samples from $\mathcal{D}_{id}$ and OOD samples from $\mathcal{D}_{ood}^{OE}$, as expressed by the following operation:

$$\text{mixup} \left((x, y) \sim \mathcal{D}_{id}, (\tilde{x} \sim \mathcal{D}_{ood}^{OE}, \mathcal{U}_c), \alpha_{idood} \right). \quad (38)$$

Empirically, *MixupOE* has shown to be particularly effective at detecting highly granular OOD samples that fall in near-ID regions of the input space. Although only previously evaluated in the natural imagery domain (e.g., detecting OOD dog and bird species), we believe that the improved fine-grained OOD potential of *MixupOE* may help significantly in the RS domain. One may argue that most OOD samples expected to be observed in the deployment environments are inherently fine-grained because the collection conditions of both ID and OOD data are identical (e.g., same sensor, same weather conditions, same background patterns, etc.).

OOD detection algorithm. The next critical detail of the classifier + OOD detector system is how we define the detector. Most OOD detectors and OSR algorithms inherit the view that the DNN is a general feature-extractor from which the detector can leverage some signal to make the ID/OOD prediction. The outlier exposure-based training schemes discussed in the previous section function by tightly controlling the DNN’s prediction confidence on ID and OOD data. So, it is natural that we employ an OOD detector that makes use of the classifier’s prediction confidence as the primary input signal. Specifically, we use a temperature-scaled maximum softmax probability detector called ODIN [18]. For input x and temperature T , the ODIN detector probes the DNN’s logit vector $g(x; \theta)$ and produces a score $S_{ODIN}(x)$ which is the maximum value from a temperature-scaled softmax vector, as described by

$$S_{ODIN}(x) = \max_{i \in \mathcal{C}} \frac{\exp(g_i(x)/T)}{\sum_{j=1}^{\mathcal{C}} \exp(g_j(x)/T)}. \quad (39)$$

$S_{ODIN}(x)$ is then compared to a pre-defined threshold β_{thresh} . If $S_{ODIN}(x) \geq \beta_{thresh}$, x is deemed to be ID. Else, x is regarded as OOD (and the system will abstain from releasing the classification prediction).

Learn-online workflow. A key novelty in this work is the ability for the classifier + OOD detector system to learn online from the environment. To allow for this functionality, we outline a workflow for such a system and identify a few additional components to consider in the design. Our proposed sequence of steps for how to train and evaluate a classifier + OOD detector system that can learn online is as follows:

- 1) Train the DNN classifier offline (e.g., with *standard*, *OE*, or *MixupOE*);
- 2) Compute β_{thresh} on a small ID-only validation set to achieve $K\%$ True Positive Rate with the OOD detector;
- 3) Deploy the system into the test environment and collect a library of N ID samples and N OOD samples (total library size = $2N$);
- 4) Allow a human observer to correct the ID/OOD labels in the collected library;
- 5) Update the DNN model using the library samples only;
 - a. (Optional) Repeat steps 3-5 if necessary/possible

- 6) Re-deploy the system into the test environment to compute the final accuracy and detection statistics.

Here, the library collection process in 3) is non-trivial and is a key novelty of this work. In our systems, we allocate a fixed memory budget for collecting the library. This means that we do not have the luxury of saving all the observed data in 3) then doing a post-processing analysis to reduce the library down to $2N$. Rather, our systems must make a decision to either keep a given sample, or discard it, on the fly. And, if the decision is to keep the sample by adding it to the library, we must evict a current library entry to make room for the new sample. In this work, we call the logic for whether to keep/discard a test sample as the library policy.

Library collection policy. In this work, our library collection policies are guided by two intuitive concepts: *hardness* and *diversity*. If a policy collects based on perceived hardness, there is emphasis on collecting ID samples that have low ID/OOD scores (i.e., ID samples that are predicted with low confidence), and OOD samples that have high ID/OOD scores (i.e., OOD data that are predicted with high confidence). Functionally, we can estimate hardness by measuring the proximity of a sample’s $S_{ODIN}(x)$ to the OOD detector’s β_{thresh} . In practice, an ID library collected with the hardness property would contain the N samples that fell closest to β_{thresh} that are also greater than or equal to β_{thresh} . Similarly, an OOD library collected with the hardness property would contain the N samples that fell closest to β_{thresh} that are also less than β_{thresh} .

If a policy collects based on diversity, there is emphasis on collecting samples from each of the different categories in both the ID and OOD distributions. Collecting diverse ID samples is straightforward, as we work to compile equal amounts of data from each of the known ID classes. However, collecting diverse OOD samples is more difficult, as we want to collect samples from different OOD categories but we don’t actually know what the OOD categories are. In this case, we estimate OOD diversity by collecting samples that are predicted as each of the known ID classes. Intuitively, if different OOD samples are predicted as different ID classes, they likely contain unique features and thus it may be beneficial to update the DNN on both types.

Note, either concept alone may not be ideal for collecting the libraries. In some settings, one specific type of OOD data may be more difficult to detect than others (e.g., this may occur if one OOD category has a particularly strong semantic relationship with one of the ID classes). In this case, using a hardness-only policy may result in the OOD library containing only one type/category of OOD. Then, when the model is updated in 5) it may overfit to this particular form of OOD data and yield no benefit to any of the other forms. On the other hand, if there is no consideration of hardness, the libraries may fill with samples that the model is already accurate on, so the effect of the update step may be minimal. For these reasons, one may also consider a hybrid approach that is guided by both hardness and diversity.

In total, we consider four library collection policies: *queue*, *hard*, *diverse*, and *hard+diverse*. The *queue* policy is the most straightforward, as it simply keeps the most

recently observed samples while evicting the oldest (i.e., first-in-firstout). The *hard* policy collects samples purely on the concept of hardness, as described above. The *diverse* policy collects samples solely on the concept of diversity, as described above. The *hard+diverse* policy collects samples using concepts of both hardness and diversity. Note, these policies are applied separately to the ID and OOD libraries.

Updating the deployed model. The final methodological detail to discuss is how to update the DNN with the data in the corrected libraries. In the spirit of transfer learning, one could treat the pretrained model as a feature extractor and only update the output classification layer weights using the new data. Another option is to perform fine-tuning of the all the pretrained weights. One could also add regularizers [23] or perform replay-based [24] or expansion-based [25] online/incremental learning. In this work we opt to perform full model fine-tuning (i.e., all weights are updated) using the same base training algorithm that was used in the offline training step (i.e., if the DNN was trained offline with *MixupOE* then we will fine-tune with *MixupOE*). With the correct learning rate setting, we find this method to work well.

4.0 RESULTS AND DISCUSSION

In this section, we cover the experimental setup, results, and analysis for each of our projects. **Section 4.1** covers the first thread of our research dealing with creating and understanding novel adversarial attack methods, whereas **Section 4.2** covers the second thread of research where we seek to improve the robustness of models to adversarial attack and anomalous data. The corresponding motivations and methodologies for these works are detailed in **Section 3.0**.

4.1. Enhancing and Understanding Adversarial Attack Effectiveness

4.1.1 Fast Generative Poisoning Attack.

As the starting point of our experiments, we choose to generate poisoned samples from two initial values, i.e., random initialization (using random noise as initial sample) and normal initialization (using normal sample as initial sample). According to **Section 3.1.1** both the numerical method and FastGPA use the loss of the attacked model w.r.t. validation samples $\sum_{i=1}^{valSize} L_i^{(a)}$. Therefore, the size of the validation set that is used to calculate $\sum_{i=1}^{valSize} L_i^{(a)}$ has a strong influence on the convergence speed of the poison sample generation and the attack effectiveness. More specifically, a larger validation set size may provide a more general consideration of the entire dataset and amortize the impact of some particular samples. Since increasing the val size increases the computation time, we choose a typical val size of 100 in our experiments and use 10 and 1000 as two references. The results are obtained using the single-sample attack.

Figure 12 compares the impacts of the val size and the initial data selection on model accuracy and loss for MNIST. The blue lines denote the testing accuracy of the attacked model (the right y axis) and the red lines denote the average loss of the poisoned model w.r.t. validation samples (the left y axis) at different iterations of a generation process. Specifically, when iteration (x axis) = 0, the corresponding dots represent the metrics of the original model. Iteration = 1 means that the model is poisoned with the starting point (here, normal initialization equals to label flipping), i.e., without further applying the poisoning algorithm. Iteration = 2 means that we apply the poisoning algorithm once to the starting point, and the resulted poisoned sample is used to poison the model. Iteration higher than 2 means applying the poisoning algorithm multiple times. The following conclusions can be drawn from the figures:

Normal initial sample is a better choice than the random one from an attacker's point of view. As can be seen from the figures, the trends of loss increase and accuracy degradation of the normal initial sample (solid lines), i.e., label flipping attack, are much steeper than the ones of the random initial sample (dashed lines), i.e., noisy sample attack. The loss (accuracy) of the normal initial sample also converges at a higher (lower)

value than that of the random initial sample. Label flipping or noisy sample attack could compromise the target model, however, with very low effectiveness. In the figures, the first point of a line (iteration 0) refers to the original result of the target model, while the second point (iteration 1) refers to the result of the model poisoned by the initial sample, i.e., label flipping attack (using normal initial sample) or noisy sample attack (using random initial sample). As we can see, the accuracy degradation at iteration 1 is limited in all scenarios. For example, the testing accuracy of iteration 1 is $\sim 70\%$ when the val size is 100. A larger val size results in a higher attack effectiveness. Comparing the accuracy degradations in the three sub-figures, we note that the curve corresponding to a larger val is more stable than that of a smaller val size. When starting from a random sample, a larger val size also results in a lower converged accuracy. However, a larger val size incurs a longer computation time. Note that the selection of learning rate plays an important role here: We conduct experiments on MNIST using different learning rates (0.1, 0.05, 0.01) and the resulted accuracies are 9.67%, 11.78%, and 87.01%, when val size = 10.

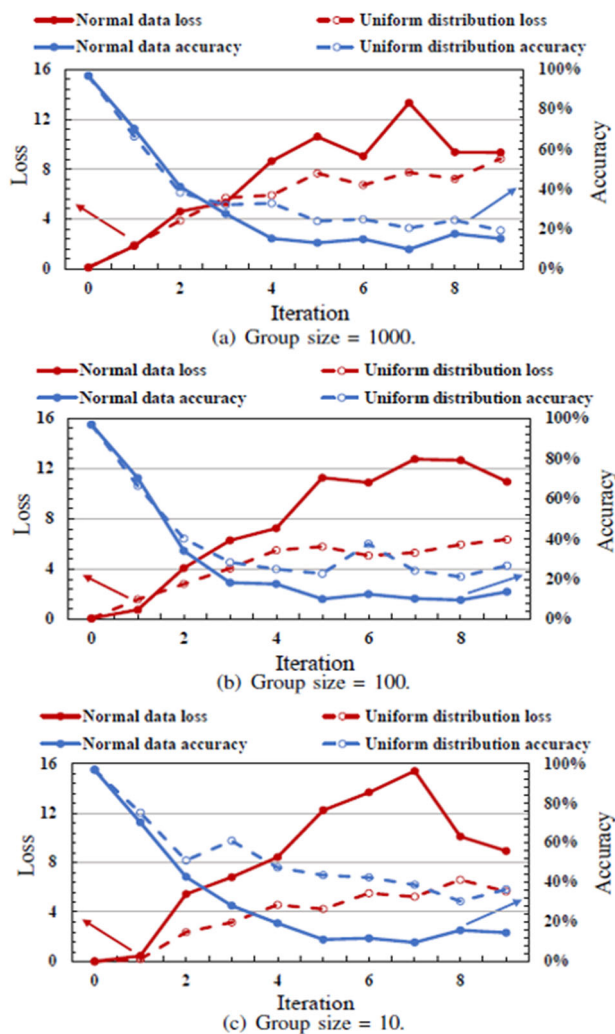


Figure 12. The trend of the loss and accuracy of the numerical gradient method under different group sizes for the MNIST dataset

Poisoned sample visualization. In Figure 13 we demonstrate the poisoned sample generation processes of some selected sample images, starting from the normal samples. For example, a normal “5” in MNIST is chosen as the initial sample while the label is set to “6” in Figure 13 (b). The generated poisoned image gradually transforms to a pattern between “5” and “6” until convergence. This observation is different from the intuitive thought that the converged pattern should be very unlike the attacking class, i.e., “6” in this example. It could possibly be explained that a pattern with a structure similar to that of the normal one but with very different features will compromise the learning process the most. The converged patterns from CIFAR-10 and ImageNet do not follow this observation, as can be seen in Figure 13 (c,d,e). This is because these two examples have much larger dimensions: $3 \times 32 \times 32 = 3,072$ on CIFAR-10 or $3 \times 224 \times 224 = 150,528$ on ImageNet vs. $1 \times 28 \times 28 = 784$ on MNIST. On the other hand, the numerical gradient method generates a noisy (or less smooth) poisoned image, which is shown in Figure 13 (a). Such a poisoned sample is easier to be detected by a low-pass filter-based mechanism.

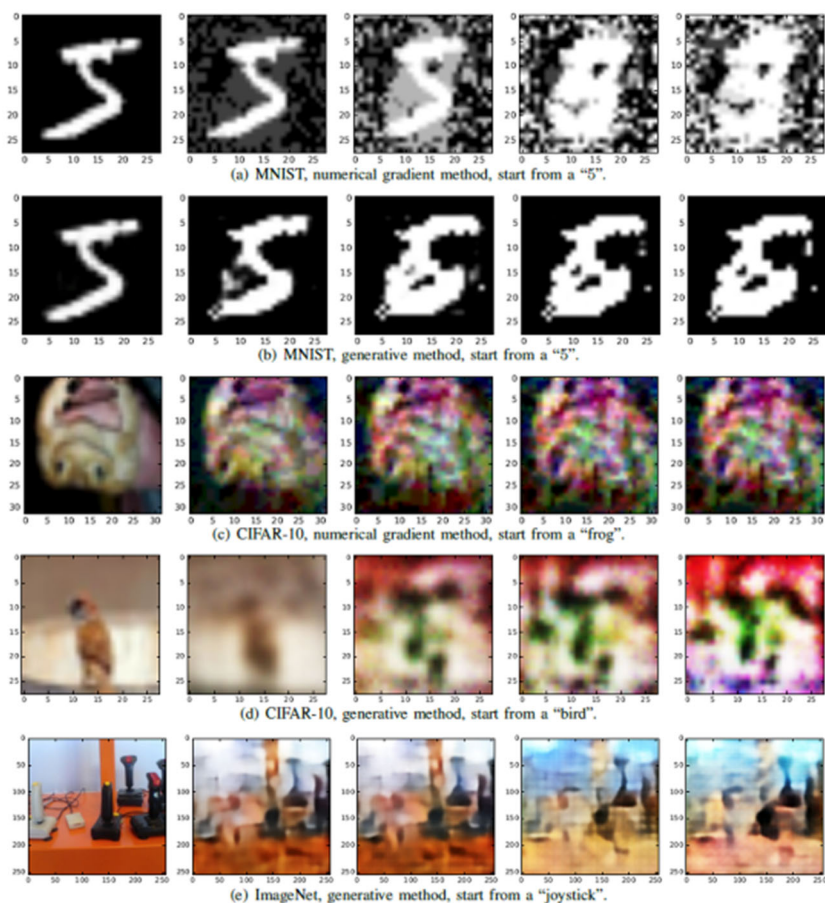


Figure 13. The process of poisoned data generation under different configurations

Numerical gradient method vs. FastGPA. Both the numerical method and FastGPA use the loss of the attacked model w.r.t. validation samples $L_{val}^{(a)}$. Therefore, the size of the validation set affects the convergence speed of the poison sample generation and the attack effectiveness. More specifically, a larger val size may provide a more general consideration of the entire dataset and dilute the impact of some particular samples. We compare the numerical gradient method and FastGPA over three datasets using the normal initialization with three val size settings (1000, 100, 10), respectively. Here, the structure of the autoencoder used as G for MNIST is 784-200-200-784. G for CIFAR-10 is a deeper fully-connected autoencoder with 6 layers, and the one for ImageNet consists of 14 convolutional and transposed convolutional layers. Table 2 compares the average computation time to generate one poisoned sample (TIME), and the least accuracy achieved by the target model (ACC.).

The improvements of FastGPA over the numerical gradient method are also presented in Table 2 (row “im”). For ImageNet, we only report the top-5 accuracy as a convention. The computation time per iteration of the numerical gradient method for ImageNet is over 24 hours so it’s not included in the table. It proves the poor scalability of the numerical gradient method on large datasets and complex models. The best result of every metric under each dataset is marked in bold. According to Table 2, the numerical gradient method always consumes a much longer time than FastGPA especially when the val size is large: for example, $\sim 240\times$ for CIFAR-10 when val size is 1000. For MNIST, FastGPA converges to a higher accuracy, i.e., 17:03% (the worst case) compared to the one of the numerical gradient methods, i.e., 9:67% (the worst case). This difference in accuracy is since we use a small network on a small dataset, and the accuracy degradation converges too fast for FastGPA. For the bigger dataset CIFAR-10, both methods converge to similar accuracy levels (the only exception is val size = 10, which has fewer validation data thus incurring a larger variance). For ImageNet, the best-case top-5 accuracy of FastGPA (at val size = 1000) degrades from 87:85% to 79:23%, while the numerical gradient method could not generate results within a reasonable execution time. It implies that FastGPA has a better scalability and works better for larger neural networks. Moreover, the computation time of the numerical gradient method has a strong dependency on the val size, which is not visually observed in FastGPA.

Table 2. The comparison of the numerical gradient method and FastGPA about computation time and accuracy under single-sample attack (NG = Numerical gradient method, GPA = FastGPA, im = improvement)

		VAL SIZE = 1000		VAL SIZE= 100		VAL SIZE= 10	
		TIME (S)	ACC. (%)	TIME (S)	ACC. (%)	TIME (S)	ACC. (%)
MNIST (1, 28, 28)	NG	345.85 $\pm$ 10.50	8.84	36.18 $\pm$ 3.13	9.64	4.18 $\pm$ 0.91	9.67
	GPA	3.88 $\pm$ 0.68	16.59	3.41 $\pm$ 0.98	17.03	3.84 $\pm$ 0.60	16.41
	im	89.14 $\times$	0.53 $\times$	10.61 $\times$	0.57 $\times$	1.09 $\times$	0.62 $\times$
CIFAR-10 (3, 32, 32)	NG	3447.8 $\pm$ 83.55	20.51	580.70 $\pm$ 17.44	20.91	259.60 $\pm$ 6.56	20.81
	GPA	14.40 $\pm$ 1.14	20.74	11.69 $\pm$ 1.46	20.88	12.56 $\pm$ 0.77	23.40
	im	239.38 $\times$	0.99 $\times$	49.67 $\times$	1.00 $\times$	20.67 $\times$	0.89 $\times$
IMAGENET (3, 256, 256)	NG [†]	—	—	—	—	—	—
	GPA	34.41 $\pm$ 0.78	79.23	23.62 $\pm$ 1.00	84.10	22.48 $\pm$ 0.63	81.59

[†] The computation time per iteration is over 24 hours so the results are not presented.

Multi-sample poisoning attack. We may observe from Table 2 that single-sample attack is less effective on ImageNet among all three datasets. To further improve the attack efficiency on ImageNet, we study the effectiveness of multi-sample poisoning attack, i.e., attacking the target model consecutively with multiple poisoned samples. We find that consistently updating G over all the poisoned samples is an effective method for multi-sample attack. Here, we do not renew G until all the poisoned samples are generated. In this way, the accuracy of the target model degrades faster than renewing G after individual generation. We also performed limited fine-tuning of G, i.e., only updating G a couple of times for each sample, to balance between generation rate and attack efficiency.

The results are shown in Figure 14. Two strategies of multi-sample attack are performed in our experiments. The first one is to randomly assign labels to the poisoned samples (represented by the solid curves), namely, the non-targeted attack. The second one is to assign a fixed label to these samples (represented by the dashed curves), namely, the targeted attack. For each strategy, we perform the attacks with different val sizes of 10, 100, and 1000, respectively. The targeted attack (in this case, we assign label 0 to the poisoned samples) generally results in a lower accuracy than the nontargeted attack and the most effective attack happens when the targeted attack is applied with a val size = 100. The accuracy generally degrades quickly when the number of poisoned samples in the attack is between 5 and 10 and converges after 20. The converged accuracy is $\sim$ 1%, which is normally satisfying from an attacker’s point of view. Note that multi-sample attack requires generating more poisoned examples and hence, aggravates the scalability issue of the numerical gradient method and makes the attack much less practical. One extreme case is that the parameters of the target model will change significantly by the time the attacker completes the poisoned sample generation (several weeks or months). In contrast, multi-sample attack using FastGPA has a much shorter poisoned sample generation time and substantially improves the practical effectiveness of poisoning attack.

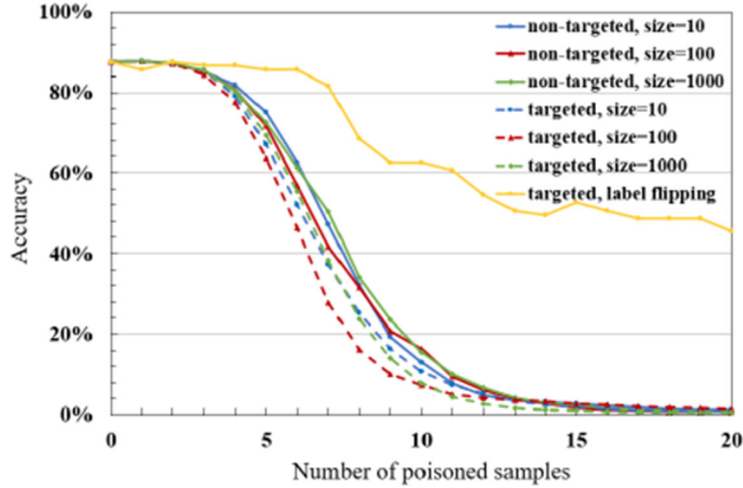


Figure 14. Multi-sample poisoning attack on ImageNet

4.1.2 Activation Attack.

To gain an understanding of whether the AA attack is feasible, we will first do testing and gather empirical results, then analyze our findings. There are two major axis of parameters to test along: epsilon and depth. Epsilon tests treat attacks from each layer separately and measure the impacts of epsilon on transferability. Testing along the depth axis involves fixing the strength of the attack and measuring attack performance as a function of which layer, we generate AA examples from. For these tests we use three baselines. The iterative targeted class method (ITCM) [26] is a targeted variant of the basic iterative method and represents a simple method. The targeted projected gradient descent (TPGD) attack with random start from [16] represents a more complex method which routinely performs between the other baselines. And the targeted momentum iterative fast gradient sign method (TMIFGSM) [5], which won the 2017 NIPS Attacks and Defences competition [27] for both targeted and untargeted attacks, represents our most complex method. The performance of each attack on the whitebox model is shown in the supplementals.

Epsilon results. The epsilon tests are a result of two experiments with a common goal, to fool a ResNet-50 classifier. First, we measure the four transferability metrics when transferring examples from DenseNet-121 to ResNet-50 (DN121 $\rightarrow$ RN50), then run a VGG19bn $\rightarrow$ RN50 test. For both, ϵ is swept as [0.0, 0.01, 0.03, 0.05, 0.07], where $\epsilon = 0$ indicates no attack. The results are shown in Figure 15. The represented AA attacks are from the best layers.

The first trend we see is that for all attacks across all metrics, as ϵ increases the attack strength increases. We also see the DenseNet-121 AA (DN-AA) is the most powerful attack, while the ITCM attacks are the least effective, and the TMIFGSM's fall in-between. At $\epsilon = 0.07$, DN-AA achieves random accuracy on the blackbox model at

91.42% error with an improvement of 7.4% over the best baseline. It outperforms the DN-TMIFGSM baseline in terms of uTR, tSuc, and tTR by 7.2%, 32.6%, and 32.5%, respectively. Also, both DN-AA and VGG-AA are higher than all baselines in terms of tSuc and tTR, indicating that both activation-based attacks are better blackbox targeted attacks than the baselines. The fact that DN-AA bests VGG-AA indicates that whitebox model architecture does affect AA performance. However, it is not intuitively surprising that the more complex DN121 model is more transferable to the RN50 blackbox model because both models are quite deep in comparison to the relatively shallow VGG.

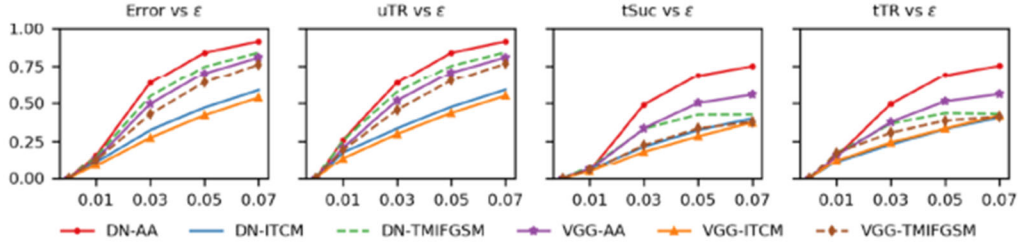


Figure 15. Transferability versus epsilon results for CIFAR-10 attacks transferring from different whitebox models to a ResNet-50 blackbox model.

Depth results. The most important results for our hypothesis are the depth results. Here, we fix $\epsilon = 0.07$ as this is where the most powerful attack was able to achieve random accuracy on the blackbox model, although our conclusions hold for all tested epsilons. We then test the AA at different depths of the models, running a full test step at each. Figure 16 shows the results of the depth sweep tests and Table 3 shows the numerical results of the most powerful layer AA attack versus baselines. In Table 3, DN and VGG are CIFAR-10 trained whitebox models, and DNIN is ImageNet trained.

The first two rows of Figure 16 are transfers from a DN121 whitebox model, and the bottom two are from the VGG19bn whitebox model. We see that layer-wise transferability characteristics are not dependent on blackbox model. Meaning, the shape of the trendlines do not change with the blackbox model. This is crucial to the feasibility of the attack because it means the attacker may use their own blackbox model to find the best transferring layer, then use that knowledge to attack the true target blackbox model. This limits the number of queries to the target model. Further, both DN121 whitebox tests show powerful transferability in the deeper layers, and both VGG19bn tests show strong transferability in the middle layers. We also see that AAs from some layers yield less powerful attacks, indicating the choice of AA layer is critical.

Table 3. Numerical transfer results (RN50 Blackbox)

Base	Attack	Error	uTR	tSuc	tTR
DN	ITCM	58.62	58.88	39.97	40.39
	TPGD	61.33	61.52	35.84	36.15
	TMIFGSM	84.12	84.32	42.53	42.90
	$AA_{L=21}$	91.49	91.48	75.13	75.38
VGG	ITCM	53.72	54.94	37.40	41.88
	TPGD	58.85	59.55	36.60	39.15
	TMIFGSM	76.19	76.75	37.72	41.14
	$AA_{L=6}$	80.81	80.97	55.64	55.98
DN_{IN}	ITCM	21.14	21.18	0.99	1.01
	TPGD	25.23	25.29	0.94	0.97
	TMIFGSM	47.75	47.76	2.25	2.25
	$AA_{L=7}$	80.58	81.77	2.57	8.63

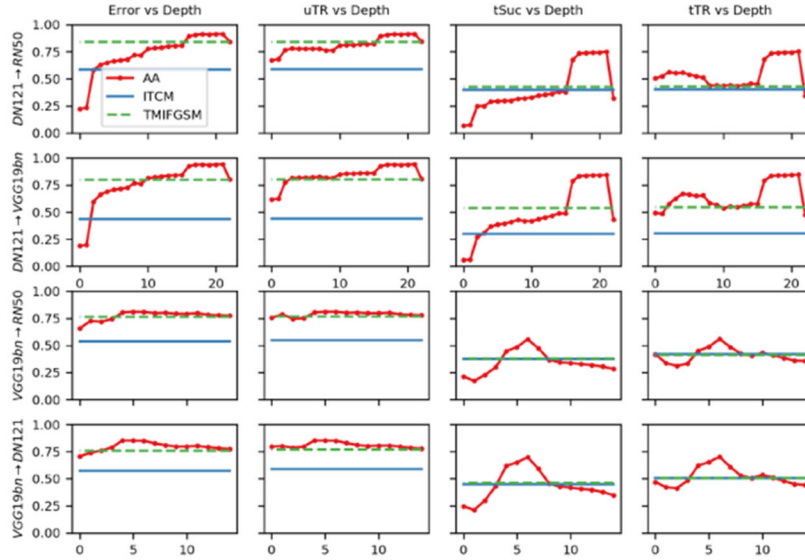


Figure 16. Error, uTR, tSuc, and tTR rates versus depth for multiple transfer scenarios; The top two rows are transfers from a DN121 whitebox model and the bottom two rows are transfers from a VGG19bn whitebox model (Dataset: CIFAR-10)

If we choose the single best layer to attack from, we can compare numerical performance to the best baseline, which in all cases is TMIFGSM. For the DN121 model the best AA is from $L = 21$, and for VGG19bn the best AA is from $L = 6$. When transferring to RN50, the DN121 $L=21$ attack outperforms the best baseline by 7.4% in error, 7.2% in uTR, 32.6% in tSuc, and 32.5% in tTR. Similarly, when transferring to RN50 from VGG19bn, the VGG19 $L=6$ attack outperforms the best baseline by 4.6%, 4.2%, 17.9%, 14.8% in error, uTR, tSuc, and tTR, respectively. Notice, for DN121 whitebox, error rate and untargeted transfer rate are both about 91.5% and targeted success and transfer rate are both about 75.2%. Such high numbers indicate that these deep and complex models have learned similar feature sets and similar decision boundary structures. Also, adversarial directions for one model are adversarial in other models. The targeted transfer metrics

further indicate that the orientation of the decision boundaries in feature space and the directions to move from one class to another in feature space are similar across models.

Finally, we can compare performance of AA attacks directly. When transferring to RN50, $DN121_{L=21}$ outperforms $VGG19_{L=6}$ by 10.6%, 10.5%, 19.5%, 19.4% in error, uTR, tSuc, and tTR, respectively. Thus, we again find that the choice of whitebox model to transfer from is important.

4.1.3 Feature Distribution Attack.

10-class ImageNet results. The primary axis of interest is how attack success rate varies with the layer depth from which the feature distributions are attacked. Figure 17 shows the transfer results between all pairs of whitebox and blackbox models. Each plot shows a metric of attack success versus relative layer depth of the generated attack. The notation $DN121 \rightarrow VGG19$ indicates adversarial examples were generated with a DN121 whitebox model and transferred to a VGG19 blackbox model.

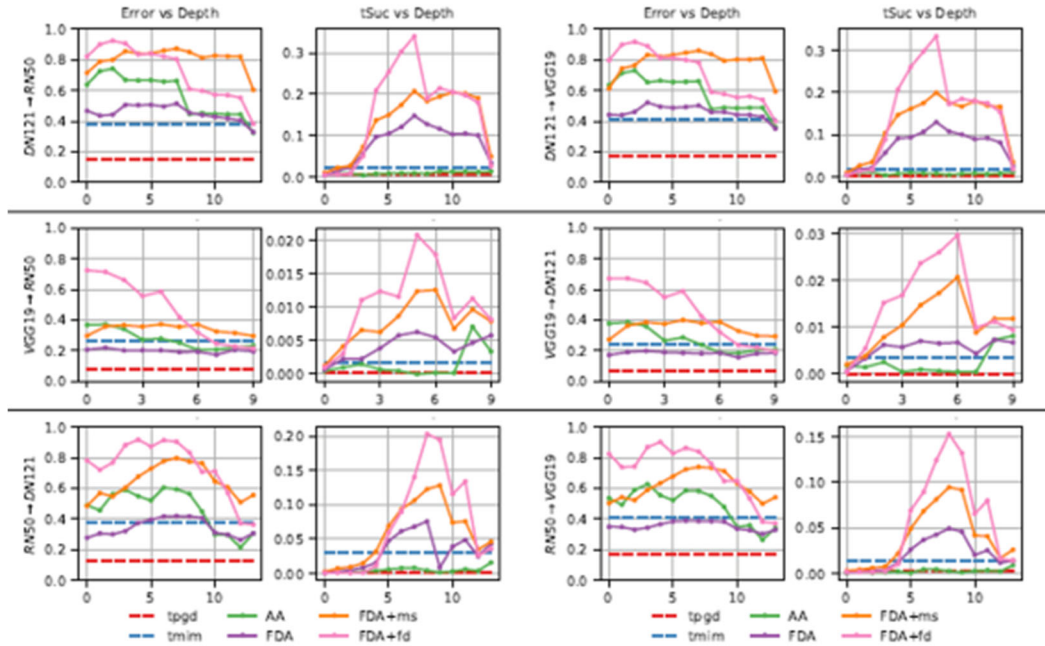


Figure 17. Targeted adversarial attack transfer results. The x-axis of each plot is the relative layer depth at which the adversarial example was generated from. Each row is a different whitebox model.

Transferability trends for FDAs from a given whitebox model appear blackbox model agnostic (e.g., the shape of the curves from $DN121 \rightarrow RN50$ are the same as $DN121 \rightarrow VGG19$). This is a positive property, as once the optimal transfer layer for a whitebox model is found, evidence shows it will be the same for any blackbox model architecture. Further, the most powerful transfers come from perturbations of intermediate features,

rather than perturbations of classification layer information. Another global trend is that in tSuc, $FDA+fd$ performs best, FDA performs worst, $FDA+ms$ is in-between, and all FDA s significantly outperform the other baselines. In the error metric, $FDA+fd$ is best in early layers, $FDA+ms$ is best in later layers, and FDA routinely under-performs the AA baseline. Although all attacks are targeted, it is relevant to report error as it is still an indication of attack strength. Also, it is clearly beneficial for the targeted attack objective to include a term that encourages the adversarial example to move far away from its starting place ($FDA+ms$ & $FDA+fd$) in addition to moving toward the target region.

We now compare performance across whitebox models. For a DN121 whitebox, $FDA+fd$ from $DN121_{l=7}$ is the optimal targeted attack with an average tSuc of 34%. For $DN121 \rightarrow RN50$, this attack outperforms the best baseline by 14% and 32% in error and tSuc, respectively. For a VGG19 whitebox, $FDA+fd$ from $VGG19_{l=5}$ is the optimal targeted attack with an average tSuc of 2.3%. For $VGG19 \rightarrow RN50$, this attack outperforms the best baseline by 15% and 2% in error and tSuc, respectively. For a RN50 whitebox, $FDA+fd$ from $RN50_{l=8}$ is the optimal targeted attack with an average tSuc of 18%. For $RN50 \rightarrow DN121$, this attack outperforms the best baseline by 27% and 17% in error and tSuc, respectively.

1000-class ImageNet results. Recall, due to the computational complexity of training one auxiliary model per class per layer per model, we ran the previous experiments using 10 randomly sampled ImageNet-1k classes. In reality, this may be a realistic attack scenario because an adversary would likely only be interested in attacking certain source-target pairs. However, to show that the 10 chosen classes are not special, and the previously identified optimal transfer layers are still valid, we train all 1000 class auxiliary models for $DN121_{l=7}$ and $RN50_{l=8}$. We exclude VGG19 because of its inferior performance in previous tests. Table 4 shows results for the four transfer scenarios. Attack success rates are all averaged over four random 10k splits and the standard deviation of all measurements is less than 1%. In these tests, for each source sample, a random target class is chosen. As expected, the 1000-class results closely match the previously reported 10-class results.

Table 4. Transferability rates for 1000-class targeted attack tests using optimal layers

attack	DN121 $\rightarrow$ VGG19		DN121 $\rightarrow$ RN50		RN50 $\rightarrow$ DN121		RN50 $\rightarrow$ VGG19	
	error	tSuc	error	tSuc	error	tSuc	error	tSuc
tpgd	23.1	0.3	21.4	0.6	20.2	0.5	22.4	0.3
tmim	48.6	1.4	45.5	2.2	44.3	2.9	46.7	1.3
FDA	64.9	15.5	64.3	18.1	56.4	12.6	54.6	6.9
FDA+ms	91.9	21.7	91.9	23.4	87.3	15.9	85.3	10.2
FDA+fd	81.2	29.0	81.7	30.9	82.6	24.3	78.9	15.9

We now investigate why a given layer and/or whitebox model is better for creating transferable adversarial examples. We also explore the hypothesis that the layer-wise transfer properties of a DNN implicate the transition of intermediate features from task/data-specific to model-specific. Intuitively, early layers of DNNs trained for

classification may be working to optimally construct a task/data-specific feature set. However, once the necessary feature hierarchy is built to model the data, further layers may perform extra processing to best suit the classification functionality of the model. This additional processing may be what makes the features model specific. We posit that the peak of the tSuc curve for a given transfer directly encodes the inflection point from task/data-specific to model-specific features in the whitebox model. Instinctively, to achieve targeted attack success, the layer at which the attacks are generated must have captured the concepts of the classes for the general task of classification, without being overly specific to the architecture. Thus, layers prior to the inflection point may not have solidified the class concepts, whereas layers after the inflection point may have established the class concepts and are further processing them for the model output.

Intermediate disruption. One way to measure why and how adversarial attacks work is to observe how the intermediate representations change because of perturbations to the input. Our trained auxiliary models afford a novel way to monitor the effects of such perturbations in deep feature space. To measure how much a layer's features have changed as a result of a (targeted) adversarial perturbation, we define layerwise *disruption* as the difference between the target class probability before and after perturbation, as measured in layer l of model f : $disruption = p(y = y_{tgt} | f_l(x + \delta)) - p(y = y_{tgt} | f_l(x))$.

Figure 18 shows the average disruption caused in each transfer scenario, using both logit-based (tmim) and feature-based ($FDA+fd$) adversarial attacks. Each row plots the disruption versus layer depth from a single whitebox model to each blackbox model (e.g., the top row results from DN121 $\rightarrow$ VGG19 and DN121 $\rightarrow$ RN50 transfers). Each line represents the average disruption caused by some adversarial attack, where all FDAs are $FDA+fd$. The first column of plots shows the impact of each attack on the whitebox model's feature distributions while the second and third columns shows impacts on the blackbox models' feature distributions.

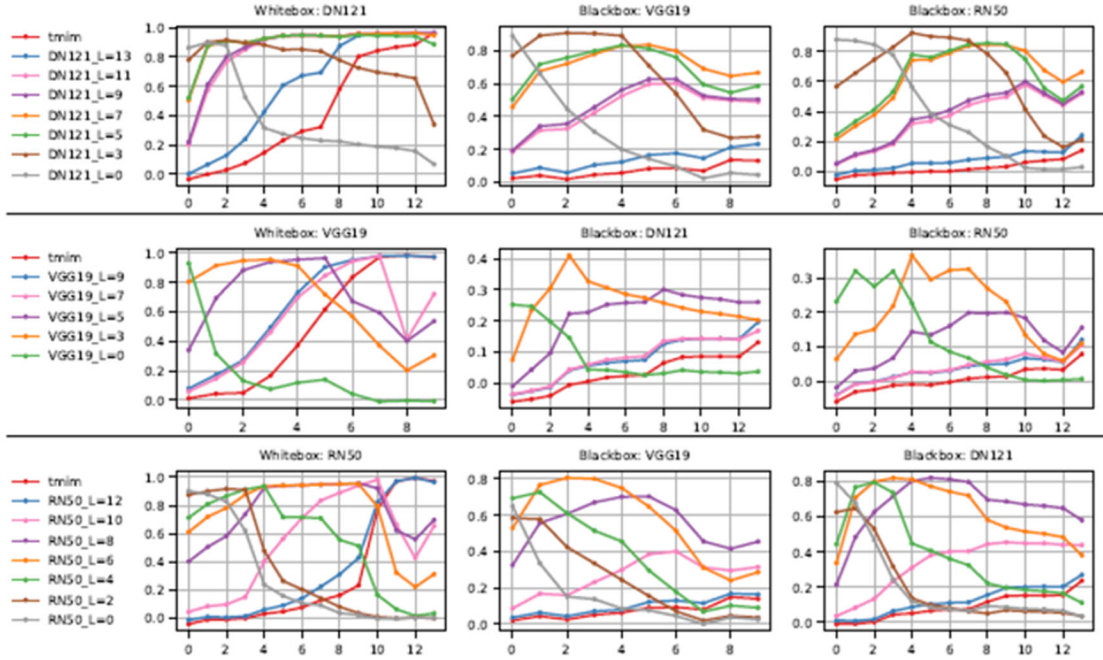


Figure 18. Disruption versus layer depth for all transfer scenarios; Each row uses a different whitebox model; Each line is a different attack where all FDAs are $FDA+fd$

It appears FDAs generated from early layers (e.g. $DN121_{l=0}$, $VGG19_{l=0}$, $RN50_{l=0}$) disrupt features the most in early layers and less so in deeper layers. Therefore, a sample resembling class y_{tgt} in an early layer does not mean it will ultimately be classified as y_{tgt} . However, recall from Figure 17 that attacks from early layers create very powerful untargeted adversarial examples (error). This indicates that early layer perturbations are amplified as they proceed through the model, just not in a class-specific manner.

Next, as expected, attacks that use information from the last layers of a whitebox model (e.g. $tmim$, $DN121_{l=13}$, $VGG19_{l=9}$, $RN50_{l=12}$) create the largest disruption in the last layers of the whitebox, but not necessarily at the last layers of the blackbox models. However, the optimal transfer attacks ($DN121_{l=7}$, $VGG19_{l=5}$, $RN50_{l=8}$) have high disruption all throughout the models, not just at the last layer. This is further evidence that perturbations of classification-layer features are overly model-specific and perturbations of optimal transfer-layer features are more specific to the data/task. Finally, notice that the maximum disruption caused in any blackbox model layer from VGG19 whitebox transfers is around 40% (row 2). For the DN121 and RN50 whitebox models, the maximum disruption is around 80%. This may explain VGG19 being an inferior whitebox model to transfer from, as the perturbation of intermediate VGG features does not in-turn cause significant disruption of blackbox model features.

Auxiliary model correlation with full model. Another point of analysis is to investigate the correlation/discrepancy between the auxiliary models at a given layer and the output of the whitebox model. This may also indicate a transition from task/data-specific to

model-specific features. We discuss discrepancy as an indication of how different the auxiliary model outputs are from the whitebox model outputs. Then correlation is the inverse of discrepancy so that when the auxiliary model outputs align well with the whitebox model outputs, the discrepancy is low and correlation is high. To evaluate discrepancy at a given layer l for input x , we aggregate the logit values (i.e. pre-sigmoid/softmax) for each class in $\mathcal{C}$, as measured by the auxiliary models $g_{l,c}$ and the whitebox model f , into separate vectors. Then, a softmax (smax) operation is performed on each vector to establish two proper probability distributions over the classes in $\mathcal{C}$. Discrepancy is then defined as the Kullback-Leibler divergence (D_{KL}) between the two distributions, or $discrep. = D_{KL} \left(\text{smax} \left([g_{l,c}(f_l(x))]_{\forall c \in \mathcal{C}} \right) \parallel \text{smax} \left([f(x)[c]]_{\forall c \in \mathcal{C}} \right) \right)$. Here, $f(x)[c]$ is the class c logit value from the whitebox model f given input x . Figure 19 shows the layer-wise auxiliary model correlations with the whitebox model outputs as measured from the average discrepancy over 500 input samples of classes in $\mathcal{C}$.

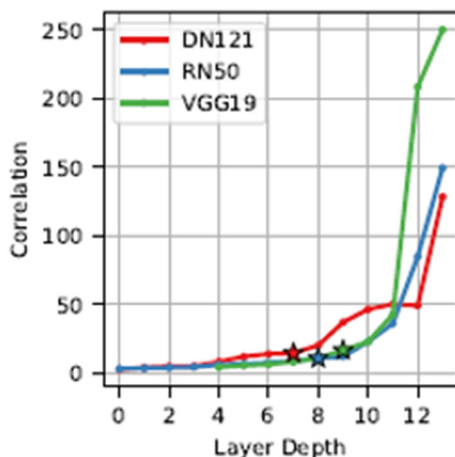


Figure 19. Correlation of a layer’s auxiliary models with the whitebox model output

Note, the shapes of the curves are more informative than the actual values. Also, VGG19 layers have been shifted in notation by +4 so that layer depth 13 is the logit layer of each model. As expected, the auxiliary models in early layers have little correlation with the model output, while auxiliary models in later layers have high correlation with the model output. Importantly, the optimal transfer layers (★) mark a transition in the trendlines after which correlation increases sharply. This effect may directly explain why layers after the optimal layer are suboptimal because the auxiliary models become highly correlated with the model output and begin to overfit the architecture. Since the auxiliary models are not highly-correlated with the output layer at the optimal-transfer-layers, we may surmise that the captured features are still mostly task/data specific.

Auxiliary model saliency. For a more qualitative analysis, we may inspect the auxiliary model saliency maps. Given an image of class y_{src} , we visualize in Figure 20 what is salient to the y_{src} auxiliary models at several DN121 layer depths using SmoothGrad

[28]. Notice, an observable transition occurs at the high-performing transfer layers from Figure 17 ($DN121_{l=5,7}$). The salient regions move from large areas around the whole image (e.g. $DN121_{l=0,3}$) to specific regions that are also salient in the classification layer ($DN121_{l=13}$). The saliency and correlation transitions together show that the well-transferring layers have learned similar salient features as the classification layer while not being overly correlated with the model output. Therefore, perturbations focused on these salient regions significantly impact the final classification without being too specific to the generating architecture.

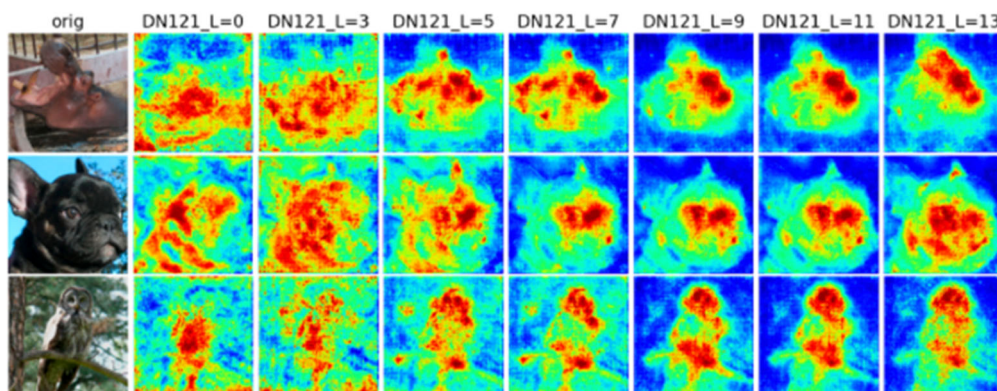


Figure 20. Saliency maps of auxiliary models on several interesting inputs across model depth

4.1.4 Multi-Layer Feature Distribution Attack.

In our analysis, we use the following ImageNet-1K models from the PyTorch Model Zoo: ResNet-34/50/101/152 (RN34, RN50, RN101, RN152); VGG16bn and VGG19bn; MobileNetv2 (MNv2); and DenseNet-121/161/201 (DN121, DN161, DN201). The notation $RN50 \rightarrow DN121$ means attacks are generated on a RN50 whitebox (source) model and transferred to a DN121 blackbox (target) model. The primary metrics of attack success are the error rate and targeted success rate (tSuc) in the blackbox model, where all clean samples are correctly classified by both the source and target models. For the attack settings, we use the common L_∞ , $\epsilon = 16/255$ and $\text{perturb_iters} = 10$.

To find the best combinations of layers for FDA(N) attacks, we use an iterative greedy search on a held-out part of the ImageNet validation set and find value in including up to 5 layers. Crucially, we have found that feature space attacks have been shown to be blackbox model agnostic: the optimal transfer layer for a whitebox model does not change for different target blackbox models. For this reason, we find the optimal layer sets for the RN50 and DN121 whitebox models only once, in the $RN50 \rightarrow DN121$ and $DN121 \rightarrow RN50$ transfer scenarios, respectively. When attacking any other blackbox model, we use the previously found layer combinations. Practically, an attacker may have two models in a sandbox environment, where one is treated as a whitebox and the other a

blackbox. They may then find the optimal layer settings in this sandbox environment, which can be used to attack any other blackbox of interest.

Transfer results in standard blackbox settings. We perform single-source single-target model transfers for $RN50 \rightarrow \{DN121, VGG16bn, RN152, MNv2\}$ and $DN121 \rightarrow \{RN50, VGG16bn, DN201, MNv2\}$. For baselines, we compare against the popular TMIM and TMIM+SGM, both of which search for adversarial directions using *only output layer information* of the whitebox model. FDA(1) is also considered a baseline. To emphasize the transferability of the noise generation technique when the blackbox model architecture may be unknown, we favor settings where the source and target model are from different architectural families to avoid potential biases.

The procedure for measuring the transfer results in Table 5 is as follows. For each source-target model pair, we randomly select 15,000 source images from the ImageNet validation set for which both models initially predict the correct source label. For each source sample, we then randomly select 5 target labels from the ImageNet label set and execute a targeted attack towards each. Thus, every table entry is an average computed over 75,000 targeted attacks. Both error rate and targeted success rate, as measured in the blackbox model, are reported as "error / tSuc".

Table 5. ImageNet transfer results (notation = error / tSuc, $\epsilon = 16/255$)

Attack \ Target	Whitebox Model = RN50				Whitebox Model = DN121			
	DN121	VGG16bn	RN152	MNv2	RN50	VGG16bn	DN201	MNv2
TMIM	44.7 / 3.0	48.7 / 1.5	41.4 / 3.2	54.9 / 0.1	46.2 / 2.3	49.6 / 1.3	44.6 / 5.5	58.5 / 1.0
TMIM+SGM	47.4 / 4.5	51.9 / 2.4	43.1 / 4.2	61.1 / 2.2	54.8 / 4.7	53.8 / 2.8	51.4 / 10.0	66.1 / 3.2
FDA ⁽¹⁾	90.3 / 19.4	86.6 / 13.6	90.8 / 17.1	84.7 / 7.1	91.8 / 20.4	90.6 / 20.6	94.8 / 35.3	88.4 / 10.7
FDA ⁽²⁾	93.1 / 28.4	91.5 / 22.9	92.7 / 22.0	89.8 / 10.7	95.1 / 22.3	94.7 / 26.3	96.6 / 38.9	93.0 / 12.4
FDA ⁽³⁾	94.1 / 31.0	93.0 / 26.1	93.2 / 22.9	91.0 / 11.8	95.1 / 22.2	94.7 / 26.0	96.7 / 37.1	92.8 / 12.6
FDA ⁽⁴⁾	94.2 / 38.1	92.4 / 30.7	93.6 / 30.6	90.2 / 14.9	92.9 / 43.7	92.4 / 42.6	96.3 / 67.5	88.9 / 20.9
FDA ⁽⁵⁾	94.2 / 38.6	92.7 / 31.0	93.5 / 29.7	90.9 / 15.1	93.8 / 44.4	93.4 / 44.4	96.6 / 68.2	90.6 / 22.4
FDA ⁽¹⁾ +xent	84.4 / 40.3	80.9 / 24.9	85.0 / 42.3	78.6 / 13.2	88.8 / 37.4	87.8 / 32.7	93.3 / 65.3	85.1 / 16.9
FDA ⁽²⁾ +xent	89.0 / 51.7	86.8 / 37.2	88.2 / 48.1	84.3 / 18.7	92.7 / 40.5	92.5 / 40.5	95.5 / 70.3	90.1 / 19.4
FDA ⁽³⁾ +xent	90.2 / 55.2	88.3 / 41.6	89.2 / 49.2	85.6 / 21.1	92.9 / 41.7	92.5 / 41.8	95.7 / 70.6	90.0 / 20.4
FDA ⁽⁴⁾ +xent	90.5 / 57.3	88.2 / 42.8	89.7 / 53.2	85.4 / 22.4	91.3 / 49.4	90.9 / 46.2	95.3 / 76.5	87.1 / 22.6
FDA ⁽⁵⁾ +xent	90.9 / 57.9	88.8 / 43.5	89.7 / 51.6	86.4 / 22.9	92.2 / 50.1	92.1 / 48.0	95.6 / 77.1	88.8 / 24.4

Our first major observation is that the +xent component alone gives large gains in tSuc. If we compare FDA(1) to FDA(1)+xent, in several cases it leads to a tSuc increase of over 20%; for example, in $RN50 \rightarrow DN121$ tSuc increases from 19.4% to 40.3%. Further, we observe +xent consistently helps, regardless of how many layers are used. The largest performance gain observed by +xent is in the $DN121 \rightarrow DN201$ case, where in the one attack layer case, tSuc increases by 30%. The next result of interest is the performance gain by adding multiple intermediate layers (FDA(N), $N > 1$). As N is increased, tSuc nearly doubles in all cases. For perspective, when we compare these results to the baseline TMIM-based methods, the targeted success rate in many cases improves by more than 10x.

We also observe that +xent and multi-intermediate-layer are complementary components: in all transfer scenarios the most powerful targeted attacks arise from using both. Specifically, consider the FDA(5)+xent attack, which is the most effective in all but one case. When VGG16bn is the target model, on average this attack outperforms TMIM+SGM by 37% error / 43% tSuc and FDA(1) by about 2% error / 29% tSuc. An interesting observation is that the +xent term harms error performance, albeit less relevant because our focus is targeted attacks, and the methods have been optimized for tSuc. However, in all cases the best FDA(N) method nearly doubles the error induced by TMIM+SGM.

Comparison to ensemble methods. Ensemble-based approaches form a separate family of methods to generate transferable adversarial examples, deviating from our previous single-source model assumption. In this setting, the attacker trains an ensemble of models (with whitebox access to each) and generates noise based on the output of the ensemble with traditional attacking techniques, e.g., TMIM. A natural iterate in ensemble methods is growing the numbers of models. We compare the effect of adding a layer within the FDA(N)+xent framework versus adding a model to the generating ensemble using the same attack settings as in the previous experiment. To avoid conflicts with models already in use, in every transfer case we use the following sequence for adding models to the ensemble: [whitebox, DN161, RN101, VGG19bn, RN34]. For example, in the RN50→DN121 transfer case, using a 3-model ensemble means generating noise from RN50, DN161, and RN101.

The results of these experiments are shown in Figure 21. The FDA methods hold a wide margin over ensemble methods in almost all cases for both error and tSuc, especially FDA(N)+xent. Also, although adding models to the whitebox ensemble tends to increase attack performance (as expected), large jumps in the ensemble’s performance usually occur when a model from the same family as the blackbox model is added. For example, in the DN121→VGG16bn case, a performance spike happens when the fourth model gets added, which is the VGG19bn model. Similarly, in the DN121→RN50 case, the ensemble method has the greatest increases when the third and fifth models are included, which are the RN101 and RN34 models, respectively.

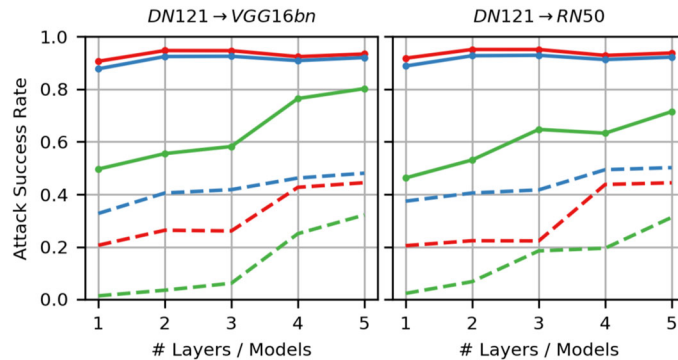


Figure 21. FDA versus ensemble attacks

4.1.5 No Label Space Overlap Attack.

Since we are not executing transfers in standard benchmark settings, we carefully design novel test environments to evaluate the efficacy and scalability of our attacks across a spectrum of realistic settings. We consider three distinct “cross-distribution” transfer scenarios. The first two, named Disjoint ImageNet Subsets (DINS) Test 1 and Test 2, are manually constructed by partitioning ImageNet into challenging subsets. The third scenario involves transfers between ImageNet and Places365 models.

DINS Test 1 and 2. To assemble the DINS Test 1 and 2 environments, we first establish 30 distinct super-classes that are comprised of five individual ImageNet classes each. For example, we create a *bird* class by aggregating the ImageNet classes [10:brambling; 11:goldfinch; 12:house-finch; 13:junco; 14:indigo-bunting], and a *big-cat* class by aggregating [286:cougar; 287:lynx; 289:snow-leopard; 290:jaguar; 292:tiger]. The super-classes are then partitioned to create the two DINS environments. Each is comprised of non-overlapping 15-class “A” and “B” splits, to be referred to as Test 1.A/B and Test 2.A/B. In our experiments, the blackbox target models in each test are trained on the “B” splits, while the attacker’s whitebox models are trained on the “A” splits. To be clear, we only intend to attack between “A” and “B” models under the same environment, i.e., Test 1.A→Test 1.B and Test 2.A→Test 2.B. We remark that the DINS tests are created to represent different difficulty levels. DINS Test 1 illustrates an intuitively more promising transfer case, because there are some obvious conceptual overlaps between the classes in Test 1.A and 1.B that may lead to natural (target, proxy) relationships. DINS Test 2 represents an intuitively harder transfer environment, where there is much less conceptual overlap between the label spaces of the whitebox (Test 2.A) and blackbox (Test 2.B).

ImageNet to Places365. Our third transfer scenario is principally created to evaluate the scalability of our attacks to more complex environments. We consider a situation where the attacker has whitebox access to ImageNet models and wishes to create targeted attacks for Places365 blackbox models. The additional complexity in this experiment comes from two primary sources. First, the sheer increase in the number of classes in both the whitebox and blackbox label spaces: ImageNet has 1000 classes and Places365 has 365 classes. Second, there is generally a finer granularity between the categories in both distributions. For example, instead of classifying at the stratum of dogs, cats, fish, etc. (as we do in the DINS tests), these more complex models have to classify between highly nuanced object sub-categories (e.g., there are over 110 dog breeds in ImageNet). From an attacking perspective, such an increase in complexity and granularity may make it more difficult to reliably target individual classes in the Places365 label space.

Figure 22 shows the class correspondence matrices for the DINS Test 1 and 2 environments. The top row of subplots is the result of forward passing Test 1.A data through the RN34, DN169, and VGG19bn Test1.B models. The bottom row is the result of forward passing Test 2.A data through the RN34, DN169 and VGG19bn Test2.B models. From these matrices, our first observation is that in both environments there exist

many strong relationships between the classes that an attacker can attempt to leverage. As discussed, we attribute this to a set of shared features between the whitebox and blackbox data distributions [7], as well as the propensity of DNNs to over-generalize [29].

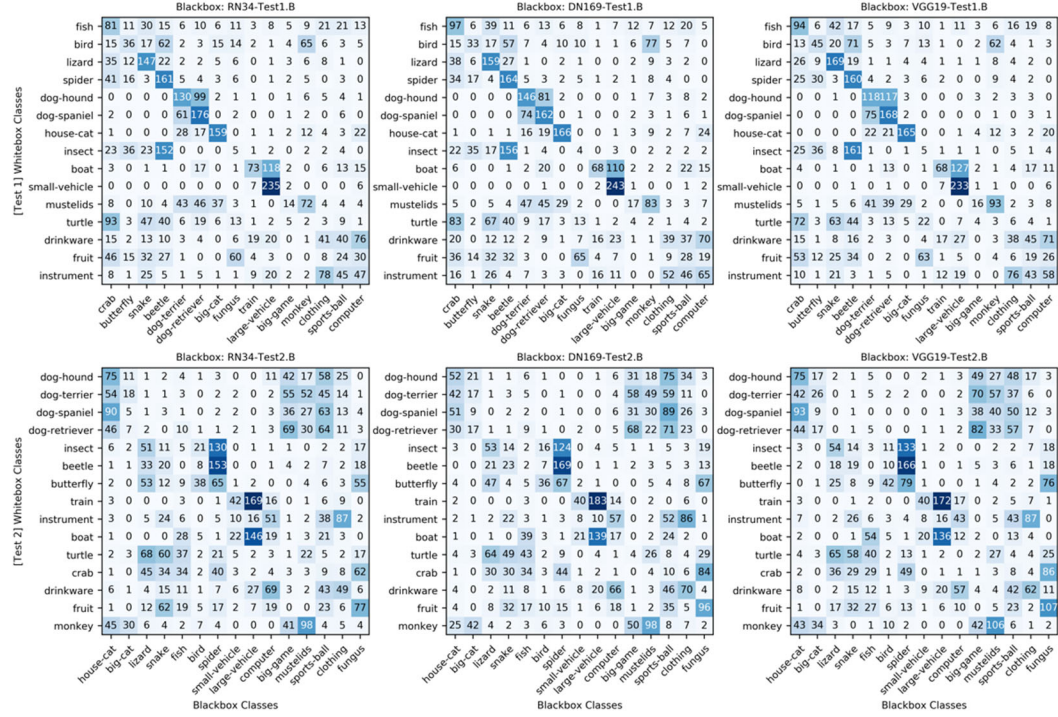


Figure 22. Class correspondence matrices for the DINS Test 1 (top row) and DINS Test 2 (bottom row) environments

Importantly, we observe the correspondences are largely intuitive. For example, the dog, cat, and vehicle classes from distributions A and B in Test 1 show significant alignment. Similarly, in Test 2, *insect* and *beetle* both correlate with *spider*, while *train* and *boat* both align with *large-vehicle*. We believe this intuitive alignment to be the result of multiple factors working together. For one, there is undoubtedly some amount of shared “low-frequency” features between corresponding classes (e.g., both *beetle* and *spider* exhibit arthropodal features such as exoskeletons and thin jointed appendages). In addition, highly correlated classes may share similar “high-frequency” textures that the blackbox model is responsive to (e.g., *lizard* and *snake* have similar scaled patterns/textures on their skin). A positive consequence of these intuitive alignments is that an attentive attacker may be able to successfully guess correspondences, even without probing the blackbox.

Another key observation from Figure 22 is that the patterns of class correspondences are predominantly a function of the data distributions and are relatively agnostic to the exact blackbox architecture. Observe that across the three sub-plots in either row of Figure 22,

the patterns of correspondences remain consistent. From an attacker’s perspective, this means that once the class-relationships between the whitebox and blackbox datasets are known, they would not have to be recomputed if/when the blackbox model architecture changes.

To measure attack potency, we consider two metrics: error-rate (error) and targeted-success-rate (tSuc). Because we enforce that all “clean” starting images are correctly classified by the target blackbox model, both error and tSuc have values in $[0, 1]$ (where numbers closer to one indicate higher success). Specifically, error measures the rate at which the blackbox model misclassifies the adversarial examples, and tSuc is the rate at which the blackbox model misclassifies the adversarial example as the specified target class. Even though our attacks are primarily optimized for tSuc, the rate of incidental errors can still be of interest to an attacker, especially because very limited options for un-targeted attacks exist in this threat model.

DINS Test 1. Table 6 shows the transfer results in the Test 1 environment for the top eleven (target, proxy) relationships. For each (target, proxy) pair, we show the results of attacking with the *TMIM* and *FDA* algorithms across the six blackbox models.

Table 6. Transfers in the DINS Test 1 environment (notation = error / tSuc)

Target (Test 1.B)	Proxy (Test 1.A)	Attack	Blackbox Models (Test 1.B)						avg.
			RN34	RN152	DN169	VGG19bn	MNv2	RXT50	
large-vehicle	small-vehicle	TMIM	25.3 / 4.3	21.1 / 3.5	25.6 / 7.1	26.0 / 6.9	30.5 / 7.3	29.5 / 8.1	26.3 / 6.2
		FDA	74.5 / 62.5	65.7 / 48.8	87.1 / 83.4	82.1 / 76.5	74.5 / 64.2	70.7 / 56.6	75.8 / 65.3
snake	lizard	TMIM	30.5 / 11.5	26.0 / 10.7	30.8 / 12.7	32.7 / 9.9	36.7 / 14.9	32.7 / 14.1	31.6 / 12.3
		FDA	63.7 / 52.2	64.9 / 56.0	79.0 / 70.6	77.3 / 47.7	68.8 / 55.8	65.1 / 56.5	69.8 / 56.5
dog-any	dog-spaniel	TMIM	23.0 / 9.0	22.7 / 10.5	31.3 / 20.5	31.9 / 18.9	29.7 / 13.4	26.9 / 10.5	27.6 / 13.8
		FDA	71.8 / 64.2	59.8 / 51.9	85.0 / 80.6	89.2 / 84.7	79.9 / 75.3	66.9 / 58.3	75.4 / 69.2
big-cat	house-cat	TMIM	29.9 / 13.1	23.1 / 7.6	29.5 / 12.5	33.2 / 13.6	33.7 / 10.7	29.5 / 10.1	29.8 / 11.3
		FDA	74.9 / 59.4	63.1 / 46.0	82.4 / 66.3	83.2 / 55.4	75.2 / 51.9	65.6 / 45.4	74.1 / 54.1
beetle	insect	TMIM	28.5 / 10.2	26.9 / 9.2	30.2 / 9.5	30.0 / 6.9	31.3 / 4.7	30.5 / 8.3	29.6 / 8.1
		FDA	66.3 / 41.9	62.6 / 40.1	76.6 / 35.2	75.1 / 28.3	65.1 / 22.1	62.5 / 24.7	68.0 / 32.0
beetle	spider	TMIM	22.6 / 3.5	20.2 / 2.5	22.1 / 2.9	26.5 / 1.7	28.7 / 1.7	26.7 / 2.6	24.5 / 2.5
		FDA	44.6 / 9.8	36.6 / 3.9	51.4 / 7.2	55.4 / 3.5	46.9 / 3.5	40.7 / 6.4	45.9 / 5.7
large-vehicle	boat	TMIM	25.2 / 0.6	19.8 / 0.5	23.1 / 0.7	26.0 / 1.6	29.1 / 1.7	27.9 / 1.2	25.2 / 1.1
		FDA	62.1 / 0.9	48.4 / 1.1	64.8 / 0.9	62.5 / 6.6	58.2 / 2.2	63.7 / 0.7	59.9 / 2.1
crab	fish	TMIM	24.3 / 4.9	22.8 / 6.0	24.8 / 7.5	27.2 / 4.9	28.0 / 6.2	28.1 / 5.4	25.9 / 5.8
		FDA	54.9 / 20.2	50.2 / 13.9	71.7 / 34.5	75.2 / 38.7	60.6 / 21.6	55.3 / 21.0	61.3 / 25.0
monkey	mustelids	TMIM	24.0 / 4.4	19.8 / 3.2	25.0 / 6.0	26.2 / 3.7	28.8 / 6.0	26.0 / 3.5	25.0 / 4.5
		FDA	60.1 / 17.7	52.4 / 10.1	69.5 / 11.0	73.4 / 21.4	68.3 / 13.8	57.0 / 7.9	63.4 / 13.7
clothing	instrument	TMIM	25.4 / 9.1	19.0 / 5.1	23.9 / 7.0	26.1 / 10.6	28.6 / 9.3	25.8 / 6.7	24.8 / 8.0
		FDA	40.8 / 12.5	33.3 / 9.3	44.4 / 6.7	46.0 / 8.9	47.4 / 11.1	43.3 / 5.1	42.5 / 8.9
crab	turtle	TMIM	27.9 / 5.3	23.4 / 6.8	28.0 / 8.0	29.8 / 7.2	31.3 / 6.7	29.1 / 5.4	28.2 / 6.6
		FDA	61.3 / 10.8	49.4 / 17.6	70.3 / 36.3	76.0 / 49.1	58.8 / 19.6	58.2 / 17.2	62.3 / 25.1

Our first observation is that across all (target, proxy) pairs, the *FDA* algorithm considerably outperforms *TMIM* in both error and tSuc. This follows our earlier intuition that attacking in feature space may be more productive than attacking at the output layer of the whitebox in this disjoint label space setting. In 7 of the 11 scenarios (as shown in bold), *FDA* achieves non-trivial average targeted transfer rates above 25%; and in 4 of these 7, the targeted transfer rates are above 50%. By comparison, some of the highest tSuc rates in standard ImageNet settings (where there is complete label space overlap) are about 50%. As such, achieving comparable transfer rates between models with disjoint label spaces is a notable result.

DINS Test 2. Table 7 shows results of attacking in the Test 2 environment for the top ten (target, proxy) relationships. Here, we simply show the average error/tSuc for each (target, proxy) pair over the six blackbox models. Many of the findings from Test 1 carry over to Test 2, starting with the observation that *FDA* consistently outperforms *TMIM* as the adversarial noise generation algorithm. In 6 of the 10 (target, proxy) pairs tested, *FDA* achieves non-trivial targeted transfer rates near or above 20%. The top transfers on average are: (*large-vehicle*, *train*) at 60.8%/44.8% error/tSuc, and (*spider*, *insect*) at 55.7%/44.4% error/tSuc. Even in the scenarios with low targeted success, the error rates are still around 50%, indicating some amount of attack potency in all settings.

Table 7. DINS Test 2 transfers (notation = error / tSuc)

Target (Test 2.B)	Proxy (Test 2.A)	TMIM-avg.	FDA-avg.
large-vehicle	train	29.5 / 8.1	60.8 / 44.8
spider	beetle	26.4 / 5.0	50.7 / 27.8
large-vehicle	boat	27.5 / 3.0	48.0 / 6.3
spider	insect	28.2 / 7.0	55.7 / 44.4
fungus	fruit	25.5 / 2.2	51.7 / 5.6
mustelids	monkey	27.3 / 8.9	58.9 / 24.3
house-cat	dog-spaniel	26.0 / 5.1	59.7 / 22.5
clothing	instrument	27.2 / 8.5	43.8 / 8.2
fungus	crab	29.4 / 2.5	55.7 / 8.8
big-game	dog-retriever	24.3 / 1.5	59.8 / 18.4

One intuitive takeaway when comparing results from DINS Test 1 and Test 2 is that the success rates in Test 2 are generally lower, especially among the top performing (target, proxy) pairs. We believe this is a natural consequence of the whitebox and blackbox data distributions in Test 2 having considerably less conceptual overlap. From our attacking perspective, such a decrease in conceptual overlap likely results in a smaller intersection between the feature sets of the whitebox and blackbox distributions for the attacks to exploit. Nevertheless, from these results we can still conclude that targeted transfer between models with no label space overlap is certainly possible, and in some cases highly effective!

ImageNet to Places365. Table 8 shows the *TMIM* and *FDA* transfer rates for the twenty (target, proxy) pairs as averaged over the three blackbox models. Note, the top section are transfers between overlapping classes and the bottom section are the non-overlapping transfers. Consistent with previous findings, *FDA* is a much more powerful noise generating algorithm than *TMIM* and is capable of inducing high error and targeted success rates for both shared and non-overlapping classes. In 9 of the 20 (target, proxy) pairs, *FDA* obtains more than 30% tSuc, including 5 cases that achieve over 55%. It is also worth noting that in 8 of the 20 transfers, *FDA*-based attacks achieve over 90% error, while the “worst” error rate amongst all twenty is still 75%. This is especially impressive, as prior work could only fool these models via query-attacks if the adversary did not have access to the Places365 dataset. Overall, these results show that transfers between large-scale DNN models trained on distinct data distributions can be very effective.

Table 8. ImageNet to Places365 transfers (notation = error / tSuc)

Target (Places365)	Proxy (ImageNet)	TMIM-avg.	FDA-avg.
83:Carousel	476:Carousel	63.5 / 4.9	93.7 / 63.0
154:Fountain	562:Fountain	60.2 / 8.9	91.0 / 71.4
40:Barn	425:Barn	58.1 / 2.0	85.1 / 20.9
300:ShoeShop	788:ShoeShop	58.0 / 3.5	94.8 / 80.8
59:Boathouse	449:Boathouse	55.3 / 1.0	86.0 / 30.6
350:Volcano	980:Volcano	54.1 / 0.7	79.5 / 32.2
72:ButcherShop	467:ButcherShop	59.3 / 1.5	90.1 / 21.3
60:Bookstore	454:Bookshop	56.9 / 2.4	90.4 / 29.6
342:OceanDeep	973:CoralReef	58.2 / 3.0	86.2 / 37.3
76:Campsite	672:MountainTent	56.0 / 2.0	88.9 / 59.5
6:AmusementArcade	800:Slot	62.7 / 5.1	92.7 / 34.9
214:Lighthouse	437:Beacon	54.1 / 0.5	76.3 / 12.7
147:FloristShop	985:Daisy	55.8 / 0.1	81.4 / 13.1
278:RailroadTrack	565:FreightCar	58.1 / 1.9	80.4 / 28.7
90:Church	406:Altar	62.3 / 0.6	93.4 / 14.1
196:JailCell	743:Prison	53.4 / 1.9	77.2 / 19.5
180:HotSpring	974:Geyser	53.2 / 0.1	83.7 / 6.6
51:Bedchamber	564:FourPoster	63.7 / 10.2	92.0 / 55.3
268:Playground	843:Swing	58.9 / 1.8	75.6 / 12.9
42:BaseballField	981:Ballplayer	54.2 / 0.1	76.6 / 22.0

4.1.6 Snooping Attacks on Deep Reinforcement Learning.

We train Deep Q Network (DQN) [30] agents on Pong, Breakout, Space Invaders, and Seaquest games via OpenAI Gym [31]. To make the environment Markovian, a state is created by stacking the last four consecutive frames, making the input volume to the models of shape 4x84x84. The proxy models use the same architecture as the small network described in [30].

To test the effectiveness of the attacks outlined in **Section 3.1.6**, we average the total reward accumulated by the target agent over 10 episodes of states perturbed by the FGM or MIFGM attacks at every time step (using the proxy gradients). We repeat this on each game for five ϵ values (i.e., attack strengths). For an upper-bound baseline attack, we use a policy-transfer attack that assumes the adversary has access to the target agent’s environment and can train an identical model to the target agent. We can then use this model as a surrogate for crafting transferable adversarial examples. Figure 23 shows the effectiveness of the transferred adversarial examples on the DQN agents.

The imitators produce the most effective attacks out of the proxy models, performing comparably to the surrogate in all games. We believe this is because these models represent the most direct proxy to the agent’s policy π_T . The assessor attacks generally require a higher ϵ to reach the performance of the baseline and imitator attacks. We hypothesize that the reason for the lower performance of the assessor attacks is because the task of predicting the exact floating-point value of a given state is a difficult objective to approximate precisely given the sparsity and variance of the reward signals.

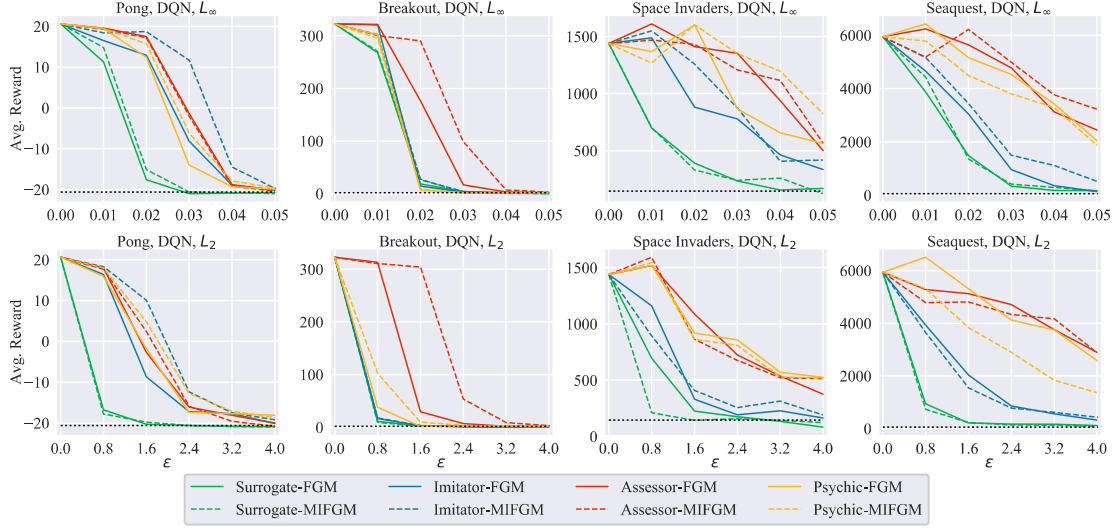


Figure 23. Performance reduction of DQN agents due to L_∞ and L_2 bounded perturbations, the black dotted line represents a random-guess policy

The attacks from the psychics perform worse than the surrogate and imitator attacks in general. This is not surprising, as the psychics operate under the most constrained threat model. In essence, the psychic model learns a noisy encoding of what the imitator learns directly. However, note that in the Breakout environment the attacks from the psychic model roughly match the effectiveness of the surrogate and imitator attacks. We believe that this is because Breakout has a deterministic state transition function (i.e., the next frame can be exactly predicted by the current frame), unlike the other games which have state elements outside of the agent’s control (e.g., hardcoded paddle in Pong, etc.). Thus, because the psychic model’s learned representation of state dynamics is a direct delineation of the agent’s policy, subsequent attacks are more transferable. This result further supports our hypothesis that proxy models that more directly approximate the target agent’s policy will yield more transferable attacks.

4.2. Enhancing Robustness to Adversarial Attacks and Anomalous Data

4.2.1 Soteria: Provable Defense Against Privacy Leakage in Federated Learning.

We evaluate our defense method against two model inversion attacks in FL. (1) DLG attack [10] assumes that a malicious server aims to reconstruct devices’ data using their uploaded gradients. In DLG attack, the server optimizes reconstructed data to minimize the Euclidean distance between the raw gradients and the gradients that are generated by the reconstructed data in back propagation. (2) Gradient Similarity (GS) attack [32] shares the similar idea with DLG. Different from using Euclidean distance in DLG, GS attack utilizes cosine similarity between the raw gradients and the dummy gradients to optimize the reconstructed data during local updates.

Defense baselines. We compare our proposed defense with two existing defense methods: (1) Gradient compression (GC) [10] prunes gradients that are below a threshold magnitude, such that only a part of local updates will be communicated between devices and the server. (2) Differential privacy (DP) [33] protects privacy with a theoretical guarantee by injecting noise to the gradients uploaded to the server. In the experiments, we separately apply Gaussian and Laplacian noise to develop two DP baselines, i.e., DP-Gaussian and DP-Laplace.

Datasets. To evaluate our defense under more realistic FL settings, we use MNIST and CIFAR10 datasets and construct non-IID datasets by following popular configurations. For each dataset, the data is distributed across 100 devices. Each device holds 2 random classes of data with 100 samples per class. By default, we perform training on CIFAR10 and MNIST non-IID dataset with 1000 and 200 communication rounds, respectively.

Evaluation metrics. (1) Privacy metric (MSE): We use the mean-square-error (MSE) between the reconstructed image and raw image to quantify the effectiveness of defenses. A smaller MSE indicates a server privacy information leakage. (2) Utility metric (Accuracy): We use the accuracy of the global model on the testing set to measure the effectiveness of FL algorithms. A smaller accuracy means a less practical utility.

Defense results: utility-privacy tradeoff. We present the results in Figure 24, where we have the following two key observations. First, when achieving the MSE such that the reconstructed image is not recognizable by humans, our method shows no drop in accuracy while the other baselines sacrifice as high as 6% and 9% accuracy under the DLG and GS attacks, respectively.

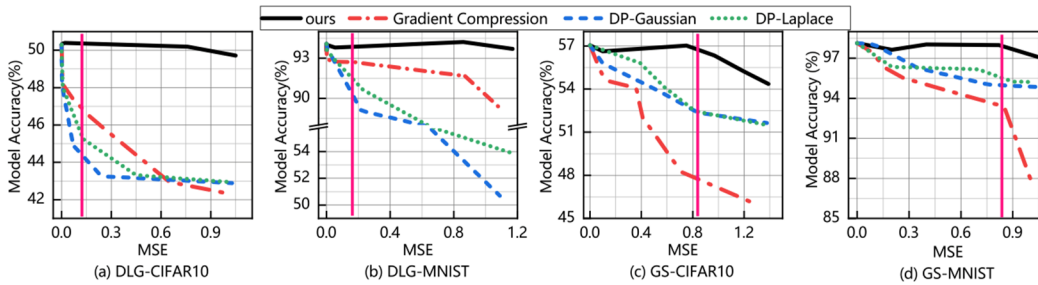


Figure 24. Compared defenses on model accuracy and MSE between reconstructed image and raw image for different attack baselines and datasets; The pink vertical line is the boundary that the reconstructed image is unrecognizable by human eyes if MSE is higher

Second, without sacrificing accuracy, our defense can achieve 160x MSE than the baseline defenses. The accuracy can be maintained by our defense until MSE being 0.8, while the baselines show significant accuracy drop with a much smaller MSE. The reason is two folded: 1) our defense does not perturb parameters in the feature extractor (i.e., convolutional layers), which preserves the descriptive power of the model; and 2) the

representations embedded in the gradients that are pruned by our defense are mostly inference-irrelevant, and hence pruning these parameters would be less harmful to the global model performance.

To perceptually demonstrate the effectiveness of our defense, we also visualize the reconstructed images. We compare our defense with GC, which is the defense baseline that also utilizes pruning. To save the space, we only show the results using the GS attack, but we have a similar observation in the DLG attack. Figure 25 shows the reconstructed image of a random sample in CIFAR10: the reconstructed image generated by our defense becomes unrecognizable when pruning only 50% – 60% parameters in the FC layer. However, when applying the GC defense, the reconstructed image is still recognizable even when 80% parameters of the whole model are pruned. Note that being unrecognizable to humans is not the goal of defense, as the private information might still reside in the image though the image is not perceptually recognizable. Nonetheless, a MSE higher than the threshold that makes the image recognizable still serves as a meaningful indicator of privacy defense.

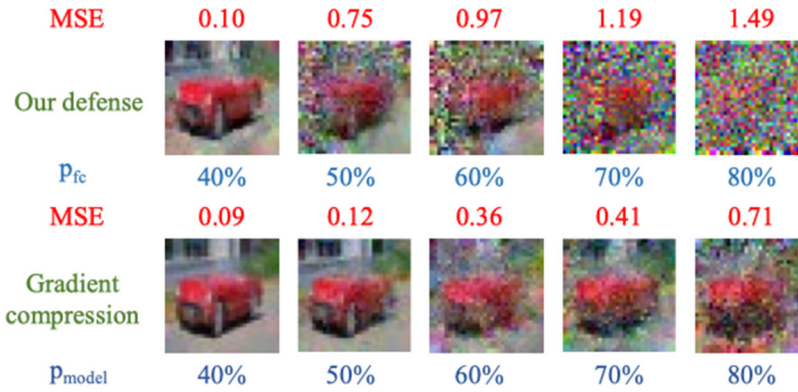


Figure 25. Comparing our defense with GC under the same pruning rate on model accuracy and MSE on a random image in CIFAR10

4.2.2 Adversarial Outlier Exposure for Reliable Operation of SAR-ATR Models.

In all following experiments, the setup is nearly identical. Regardless of the five different training procedures, all models are trained with Gaussian noise augmentation, use the ResNet18 model architecture and input 64×64 pixel crops of the SAMPLE images. Every model is trained with a specified $[K, J]$ setting, and for each configuration, we run 100 iterations. Within a single iteration, we randomly initialize the model parameters and select a random set of J classes to holdout from the training set. We then train the model and evaluate the detection statistics for the specific holdout dataset (and any other OOD dataset of interest). All final reported statistics are averaged over the 100 iterations. Finally, every model is trained for 60 epochs using the ADAM optimizer and has an initial learning rate of 0.001 and a single learning rate decay step at epoch 50 to 0.0001.

We do not use a validation set, so to measure accuracy and detection statistics, we simply take the final model after 60 epochs of training.

The first primary experiment is to measure ATR performance using the J classes of holdout data. We consider performance as a function of detection rates on the OOD set and accuracy on the ID task. We also measure the impact of holdout class choice. To quantify detection ability across a variety of operating conditions we measure how the AUROC and TNR@95TPR detection metrics change with K , J , and training algorithm. Table 19 shows results for values of $K \in \{1, 0.5, 0\}$, $J \in \{1, 2, 3\}$, and all five training algorithms (for the AT-based methods, we include two levels of perturbation). First, we notice that the value of K significantly affects the performance. It appears markedly easier for models trained at $K = 1$ and $K = 0.5$ to detect the holdout OOD data, as compared to the $K = 0$ models. As evidence, the average AUROC values for the standard trained models are 92, 88, 72 for values of $K = \{1, 0.5, 0.0\}$, respectively. This is a somewhat expected result, as the $K = \{1, 0.5\}$ models can learn high-quality features from the measured data distribution during training, which evidently makes them better equipped to detect measured OOD samples during evaluation. The $K = 0$ models learn exclusively from the synthetic data representations, so for the detection task such models must also contend with the existing distribution gap between the synthetic and measured datasets.

Table 9. Detection results for different model training algorithms and combinations of K and J (Notation = AUROC / TNR@95TPR)

Training Method	$K = 1$			$K = 0.5$			$K = 0$		
	$J = 1$	$J = 2$	$J = 3$	$J = 1$	$J = 2$	$J = 3$	$J = 1$	$J = 2$	$J = 3$
standard	92.5 / 75.2	91.4 / 72.2	92.3 / 74.4	88.9 / 60.9	87.7 / 56.8	89.3 / 59.8	72.8 / 26.5	71.7 / 22.9	72.4 / 23.4
lblsm	97.6 / 88.3	97.7 / 87.5	97.8 / 87.8	95.2 / 73.1	94.5 / 71.4	95.1 / 73.5	76.1 / 22.9	77.3 / 24.2	76.9 / 24.0
AT ($\epsilon = 2$)	94.6 / 81.3	92.3 / 75.6	91.6 / 74.0	90.2 / 68.8	89.2 / 65.0	90.7 / 66.4	74.7 / 30.4	72.5 / 22.8	73.5 / 24.8
AT ($\epsilon = 8$)	96.0 / 84.5	94.0 / 81.6	93.8 / 80.4	92.8 / 77.1	91.7 / 74.1	92.5 / 74.6	73.4 / 30.7	71.8 / 25.8	73.0 / 27.9
OE	97.0 / 86.2	96.2 / 82.3	96.3 / 83.2	93.5 / 70.7	93.1 / 68.0	93.1 / 71.1	80.6 / 31.6	79.1 / 32.7	80.5 / 34.3
AdvOE ($\epsilon = 2$)	98.6 / 93.5	98.0 / 90.5	98.2 / 91.4	96.4 / 83.0	95.8 / 80.7	96.1 / 82.0	82.1 / 39.6	81.0 / 36.6	81.6 / 39.2
AdvOE ($\epsilon = 8$)	99.2 / 96.9	99.0 / 95.6	99.0 / 95.8	97.5 / 89.1	97.8 / 89.4	97.7 / 90.1	82.9 / 41.6	81.3 / 39.9	83.7 / 45.2

Next, it appears that within a specified value of K , the value of J does not drastically impact the performance. In this work, we only consider values of J from 1 to 3 because we wish to focus on how the detection rates change with the training method and not how detection rates are affected by extremely low training data counts. We believe that for higher values of J , our training parameter choices (i.e., model architecture and levels of Gaussian noise and Dropout) may be suboptimal, which will inadvertently affect detection rates. We leave the study of training dataset size versus the detection performance as a future work.

Finally, we look across training procedures, where we consider the standard model as the baseline for comparison. Before discussing the implications of OE, we note that across all

combinations of K and J , training with AT ($\epsilon = 8$) leads to detection improvements over the standard model. On average, AT ($\epsilon = 8$) improves AUROC / TNR@95TPR by: 2.5 / 8.2 for $K = 1$; 3.7/16.1 for $K = 0.5$; and 0.4/3.8 for $K = 0$. We postulate that the quality of the learned robust features is a driving reason for the improved detection performance of the AT ($\epsilon = 8$) model. In all but one case, lblsm also improves the performance over the standard model by similar margins. When vanilla OE is included, the models are also better than the standard model across all combinations of K and J . However, in most cases OE performs only slightly better than the AT ($\epsilon = 8$) model, and in the $K = 0.5$ setting, OE underperforms AT ($\epsilon = 8$) in the TNR@95TPR statistic. Lastly, we see that our AdvOE ($\epsilon = 8$) training procedure is the top performer in all cases. On average, AdvOE ($\epsilon = 8$) improves AUROC / TNR@95TPR over the standard model by: 7.0 / 22.1 for $K = 1$; 9.0/30.3 for $K = 0.5$; and 10.3/17.9 for $K = 0$. Its margins over the vanilla OE model are also considerable, meaning that the addition of the inner maximizers proves substantive.

Besides detection ability, accuracy on the ID test sets is a key priority. Table 10 shows the average classification accuracy of the $K = 0$ models on the 10 – J class measured test datasets. All the $K = 1$ and $K = 0.5$ models are near or above 99% accurate, so we do not include them here. Interestingly, the primary observation from Table 10 is that our AdvOE training procedure is also capable of yielding the most accurate models. On average, the AdvOE ($\epsilon = 2$) models are over 1.1% more accurate than the standard and 0.97% more accurate than the OE models, across values of J . We also see that unlike in the detection results, which preferred $\epsilon = 8$, in the accuracy results the AdvOE ($\epsilon = 2$) models are slightly better. This finding shows that AT models with large ϵ values can yield slight accuracy degradation but can induce significant robustness advantages.

Table 10. Accuracy of $K = 0$ models for each training algorithm

Training Method	$K = 0$		
	$J = 1$	$J = 2$	$J = 3$
standard	92.56 ± 1.64	92.19 ± 2.61	92.68 ± 3.08
lblsm	93.37 ± 1.58	92.85 ± 2.12	93.21 ± 2.24
AT ($\epsilon = 2$)	93.23 ± 1.42	93.05 ± 2.48	93.43 ± 2.41
AT ($\epsilon = 8$)	92.47 ± 1.68	92.19 ± 2.47	92.89 ± 2.69
OE	92.70 ± 1.88	92.29 ± 2.87	93.04 ± 2.76
AdvOE ($\epsilon = 2$)	93.47 ± 1.54	93.64 ± 2.30	93.83 ± 2.31
AdvOE ($\epsilon = 8$)	93.06 ± 1.56	93.34 ± 2.04	93.96 ± 2.46

Although it may seem counter-intuitive that including OOD data in training would help accuracy on the ID task, we believe that in a similar spirit to semi-supervised learning theory, there is information in the unlabeled data that the model can leverage to improve the performance on the classification task. For example, a standard model may learn overgeneralized and non-robust features of D_{id} that are also present in D_{ood}^{OE} , and thus,

would clearly not be suitable for robust classification. So, training with samples from $\mathcal{D}_{ood}^{OE}$ would force the model to learn higher quality/more-robust features of the ID classes that may in-turn improve generalization.

4.2.3 Improving Out-of-Distribution Detection by Learning from the Deployment Environment.

Test environments. The important RS task we consider is scene classification in EO satellite imagery. To our knowledge there does not exist any well-motivated OOD detection benchmarks in this domain that present both a challenging and high-resolution multi-class classification task and multiple levels of granular/difficult OOD inputs. For this reason, we opt to construct our own benchmark test environment that prioritizes these attributes. We refer readers to our paper for subtle details of the environment construction. Essentially, we partition the base dataset MLRSNet [34] into three splits: MLRSNet-ID, MLRSNet-Hard, and MLRSNet-Holdout. The first one defines the ID classes that make up the training distribution $\mathcal{D}_{id}$. The latter two are considered test OOD sets.

The next important detail is to define the $\mathcal{D}_{ood}^{OE}$ distribution for outlier exposure-based training. As mentioned, we do not use any samples from the expected OOD test distributions of MLRSNet-Hard and MLRSNet-Holdout. Instead, we look for a large, diverse, and unlabeled dataset from which our models can learn the general concept of OOD during offline training. In this work, we use the Functional Map of the World (fMoW) dataset [35] as $\mathcal{D}_{ood}^{OE}$ in the scene classification test environment. See Figure 26 for visual comparison of the samples coming from different distributions.

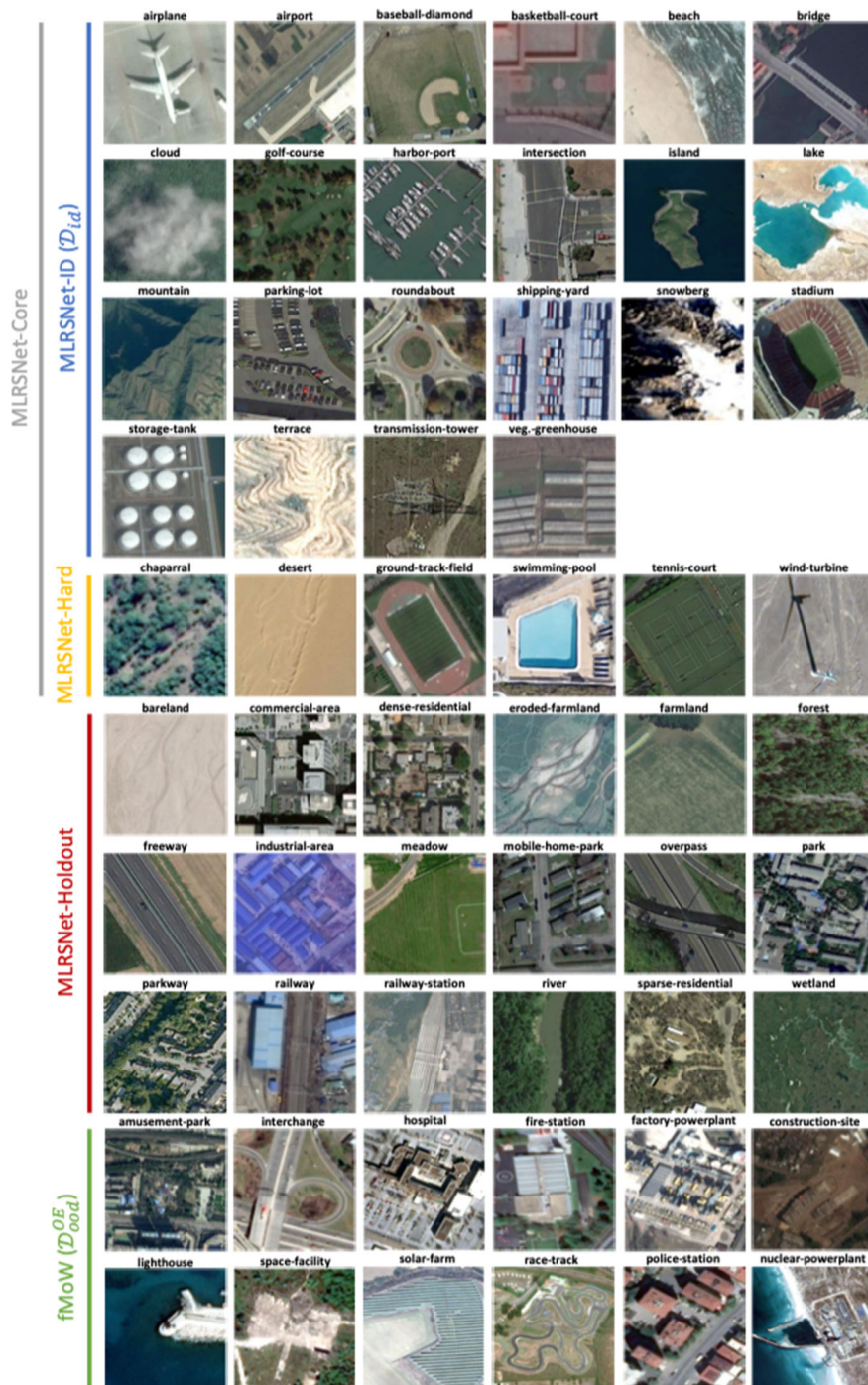


Figure 26. (Top 8 rows) Example images from each class of the MLRSNet dataset, organized by their partition into the MLRSNet-ID, MLRSNet-Hard and MLRSNet-Holdout splits; (Bottom 2 rows) Example images from some classes of the fMoW dataset which are used during outlier exposure training

Lastly, to cover a wider spectrum of data and potential OOD scenarios, we include three additional OOD datasets for testing: RSI-CB256 [36], DIOR [37], and Places365 [38]. Both RSI-CB256 and DIOR are EO satellite imagery datasets that incorporate a variety of additional scene types, including ones that do not appear in MLRSNet (e.g., pipeline, mangrove, and expressway-toll-station). On the other hand, Places365 represents a “far-away” form of OOD to gauge the system’s performance at detecting categories that are obviously not ID (e.g., physics-laboratory, florist-shop, and cafeteria). Note, these additional datasets do not exist in the deployment environment for this test, and our systems will only be evaluated on them. See Figure 27 for samples from each of these OOD sets.



Figure 27. Example images from the RSI-CB256, DIOR and Places365 datasets, which are used as additional OOD sets during testing in the EO scene classification environment.

Results. The main performance results of the EO scene classification experiment are shown in Table 11. To collect these results, we use an ID library policy of *queue* and an OOD library collection policy of *hard+diverse*. We first make a few general observations from Table 11. To begin, we find that *MixupOE* is the top performer among the (offline) models in all measurements, and in most cases is significantly better than the other two methods. We believe this is a direct result of *MixupOE* having superior coverage in the “near-ID” regions of the input space, where the domain relevant satellite image OOD samples fall. Next, notice that for the offline-trained models the MLRSNet-Hard samples are harder to detect than the MLRSNet-Holdout samples. This confirms that our methodology for constructing a difficult OOD task based on class-correlations was successful. Finally, we see that the ID accuracy of the models after update slightly decreases. However, the degradation is significantly less in the *MixupOE* models, and as library size increases most of the accuracy loss is recouped.

**Table 11. Main MLRSNet accuracy and detection results in a mixed environment
(detection notation = AUROC / TNR@95TPR / AUPR)**

Training		ID Acc.	Deployment Environment			RSI-CB256	DIOR	Places365
			MLRSNet-Hard	MLRSNet-Holdout				
standard	(offline)	98.1	82.8 / 49.8 / 70.4	93.9 / 72.5 / 89.0	95.8 / 76.7 / 94.1	74.3 / 45.8 / 60.9	94.9 / 75.9 / 91.4	
	(offline)	98.1	85.2 / 57.1 / 73.4	93.9 / 66.7 / 90.9	96.4 / 80.2 / 95.1	84.5 / 53.7 / 73.3	99.3 / 97.3 / 99.1	
OE	N=88	94.5	87.3 / 45.0 / 83.9	89.7 / 44.3 / 89.9	92.6 / 55.4 / 93.2	85.4 / 40.5 / 82.0	97.2 / 85.2 / 97.5	
	N=132	93.8	86.0 / 43.0 / 81.2	88.2 / 39.8 / 88.4	92.1 / 52.2 / 92.9	83.9 / 37.8 / 79.4	97.0 / 82.8 / 97.4	
	N=264	92.2	89.4 / 45.3 / 90.4	90.3 / 46.2 / 91.4	91.7 / 50.3 / 93.1	88.9 / 46.7 / 88.5	94.6 / 66.7 / 95.4	
	N=506	93.8	93.3 / 61.9 / 94.0	93.5 / 59.5 / 94.5	93.5 / 57.0 / 94.8	92.1 / 58.7 / 91.6	91.6 / 52.3 / 91.5	
	N=1012	95.4	95.2 / 71.6 / 95.9	95.5 / 72.7 / 96.3	95.2 / 71.6 / 96.0	93.2 / 64.5 / 93.2	93.7 / 65.7 / 93.1	
	N=2002	96.2	96.8 / 82.3 / 97.3	96.9 / 82.7 / 97.4	96.5 / 80.0 / 96.7	94.5 / 71.5 / 94.3	94.4 / 72.7 / 93.2	
	N=4004	96.9	98.2 / 90.7 / 98.4	98.0 / 89.4 / 98.2	97.7 / 87.4 / 97.9	95.4 / 76.2 / 94.9	96.4 / 79.8 / 96.8	
MixupOE	(offline)	98.6	90.4 / 67.8 / 88.1	93.9 / 76.7 / 93.1	97.9 / 91.8 / 97.8	88.1 / 65.2 / 84.3	99.4 / 98.0 / 99.2	
	N=88	97.8	91.9 / 68.1 / 91.1	94.3 / 74.6 / 94.1	97.1 / 87.6 / 97.4	91.0 / 65.2 / 90.0	96.1 / 89.9 / 93.1	
	N=132	97.5	93.5 / 71.5 / 93.3	94.8 / 76.3 / 94.7	96.6 / 86.0 / 96.8	92.7 / 69.4 / 92.1	94.3 / 87.4 / 90.7	
	N=264	97.3	94.9 / 74.1 / 95.1	95.6 / 77.6 / 95.7	96.7 / 85.0 / 96.9	94.0 / 70.8 / 94.2	94.7 / 86.5 / 91.3	
	N=506	97.1	97.1 / 83.6 / 97.5	96.6 / 81.9 / 96.9	97.1 / 84.6 / 97.5	96.1 / 79.2 / 96.5	97.0 / 85.1 / 97.0	
	N=1012	97.8	97.7 / 87.6 / 98.0	97.4 / 87.3 / 97.7	97.9 / 90.6 / 98.2	96.4 / 80.5 / 96.6	97.3 / 88.6 / 96.5	
	N=2002	97.9	98.4 / 92.1 / 98.5	98.1 / 91.1 / 98.2	98.3 / 92.6 / 98.5	97.2 / 85.6 / 97.4	96.9 / 90.6 / 94.7	
	N=4004	98.3	98.7 / 94.4 / 98.8	98.5 / 93.7 / 98.6	98.6 / 94.2 / 98.7	97.5 / 88.2 / 97.6	98.0 / 92.5 / 97.1	

Focusing on detecting OOD samples from the deployment environment, the next important observation is that in general, as library size N increases, the OOD detection performance increases. However, there is an important distinction to make here between *OE* and *MixupOE*. At small library sizes $N \leq 500$, the *OE* models show little-to-no benefit at detecting OOD data from MLRSNet-Hard/Holdout as compared to the offline model. In some statistics, performing the learn-online updates to the *OE* models harms detection performance. However, as the library size grows to $N > 500$ we begin to see significant gains from the updates. Critically, this drop in performance does not occur with the *MixupOE* models. Even at the smallest $N = 88$ library size we observe benefits to updating the *MixupOE* models, which are followed by consistent and sizeable gains as N grows. Evidently, *MixupOE* can more efficiently learn from the collected samples, especially with small libraries. Our primary takeaway from this experiment is that regardless of which library size is feasible for your system (which may be limited by the human-in-the-loop component or memory constraints), one may expect to see sizable gains in detection performance by executing the learn-online workflow with *MixupOE*.

The next observation is slightly more nuanced but uncovers an important behavior of the learn-online process. Notice how at small library sizes there are more significant gains in detecting MLRSNet-Hard samples versus gains in detecting MLRSNet-Holdout samples. Since we are using a *hard+diverse* collection policy for the OOD library, it follows that a small OOD library would be majority composed of -Hard samples. Updating the model with such a library would then intuitively lead to more substantial gains on the MLRSNet-Hard distribution. However, also notice that when the library sizes are “large enough” (e.g., $N = 1012$), the detection performance of MLRSNet-Hard and -Holdout is approximately the same, and increasing the library size further leads to nearly equal gains in detecting samples from either dataset. This can be thought of as a self-leveling behavior, where if the library size is small the focus is almost purely on the hardest

samples, then as library size increases the detection performance across all OOD-types in the deployment environment converges to about the same level.

Lastly, we observe how the learn-online workflow impacts the detection results of the three OOD sets that are not part of the deployment environment. For the RSI-CB256 and DIOR sets that are in the domain of EO satellite imagery, at larger library sizes we see gains in detection performance. Admittedly, at smaller library sizes there is a performance degradation on these sets, but it does not escalate to the level of catastrophic. For the Places365 detections, at all the library sizes we test there is an observed performance degradation, but again it does not reach the level of catastrophic. Finally, we point out that the *MixupOE* models exhibit less of this forgetting behavior, and any drops in performance are considerably milder than those observed in the *OE* models.

Analysis: Library collection policy. A critical part of the learn-online workflow is the library collection step. Because the system we consider only has a fixed memory budget for storing the library samples, it must make decisions on the fly to either save a test sample or discard it. Methodologically, we identified four policies for collecting the libraries that each employs a different approach to this keep/discard decision: *queue*, *hard*, *diverse*, *hard+diverse*. We also alluded to the idea that policy choice may be impacted by conditions of the operating environment.

In this analysis, we evaluate two unique and representative “environments” with conditions that may influence the policy selection. The first is an “ordered” environment, which represents an operational scenario where the ID and OOD data are not observed concurrently. Instead, the system is first exposed to all of the ID data, followed by all of the OOD data. This may emulate a setting where the ID data are often co-located as to be observed in pockets, followed by extended periods where only OOD data is encountered. In some ways, this is a worst-case scenario for collecting a library because without the correct policy (e.g., a queue), and using an imperfect detector, it is possible that the libraries only contain OOD data at the conclusion of 3). The second condition is a “mixed” environment, which represents a scenario where ID and OOD data are observed concurrently. This means that at any given time it is approximately equally likely that an input is either ID or OOD. We believe this may be a more normal operating scenario, and thus used this assumption in the collection of Table 12.

Table 12. Analyzing the impacts of ID and OOD library collection policy in different environmental conditions at different library sizes (Notation = AUROC-of-detecting-MLRSNet_Hard / AUROC-of-detecting-MLRSNet_Holdout)

Environment: ORDERED						Environment: MIXED					
	ID Policy	queue	OOD Policy				ID Policy	queue	hard	OOD Policy	
			hard	diverse	hard+diverse					diverse	hard+diverse
N=132	queue	- / -	89.3 / 92.7	- / -	89.6 / 93.0	queue	91.1 / 93.3	92.8 / 94.5	90.9 / 93.5	93.5 / 94.8	
	hard	88.5 / 92.7	90.9 / 93.4	89.7 / 93.5	91.3 / 92.9	hard	90.4 / 93.1	91.2 / 94.1	91.0 / 93.4	91.4 / 93.4	
	diverse	88.6 / 92.1	90.3 / 93.6	89.1 / 91.6	90.6 / 93.7	diverse	90.6 / 92.7	92.4 / 94.3	91.9 / 93.2	91.7 / 94.4	
	hard+diverse	88.8 / 92.6	91.1 / 93.3	89.3 / 93.0	91.8 / 93.7	hard+diverse	90.5 / 93.0	92.0 / 93.5	90.2 / 93.0	92.3 / 94.1	
	ID Policy	queue	OOD Policy				ID Policy	queue	hard	OOD Policy	
			hard	diverse	hard+diverse					diverse	hard+diverse
N=264	queue	- / -	89.7 / 93.5	- / -	90.0 / 93.3	queue	92.9 / 94.8	94.6 / 95.6	92.4 / 94.0	94.9 / 95.6	
	hard	89.4 / 92.9	92.2 / 94.1	90.0 / 93.6	93.0 / 94.7	hard	91.5 / 93.7	92.3 / 94.2	91.4 / 94.0	92.7 / 94.2	
	diverse	88.6 / 92.3	91.6 / 94.3	90.0 / 93.3	92.3 / 94.5	diverse	93.3 / 94.8	94.8 / 95.6	93.9 / 94.8	94.2 / 95.3	
	hard+diverse	89.7 / 93.5	92.4 / 94.1	90.3 / 93.9	93.9 / 94.4	hard+diverse	91.7 / 94.0	92.7 / 94.3	91.8 / 94.2	93.1 / 94.5	
	ID Policy	queue	OOD Policy				ID Policy	queue	hard	OOD Policy	
			hard	diverse	hard+diverse					diverse	hard+diverse
N=506	queue	- / -	88.9 / 93.7	- / -	90.4 / 93.5	queue	97.0 / 96.8	96.3 / 96.2	96.3 / 95.9	97.1 / 96.6	
	hard	89.6 / 93.2	94.5 / 95.1	90.8 / 94.2	94.9 / 95.6	hard	92.7 / 94.1	94.6 / 95.1	93.3 / 94.8	95.1 / 95.4	
	diverse	89.2 / 92.4	94.3 / 95.0	90.4 / 93.1	94.0 / 94.5	diverse	96.0 / 96.2	96.7 / 96.6	96.1 / 96.2	96.5 / 96.5	
	hard+diverse	92.2 / 94.6	95.3 / 95.6	93.2 / 95.5	95.5 / 95.8	hard+diverse	94.0 / 94.9	95.4 / 95.6	95.1 / 95.6	95.4 / 95.7	
	ID Policy	queue	OOD Policy				ID Policy	queue	hard	OOD Policy	
			hard	diverse	hard+diverse					diverse	hard+diverse
N=1012	queue	- / -	90.5 / 93.7	83.7 / 89.7	92.2 / 94.2	queue	97.4 / 97.1	97.4 / 97.2	97.4 / 97.4	97.7 / 97.4	
	hard	93.3 / 95.0	96.3 / 96.5	94.6 / 96.6	97.0 / 97.0	hard	95.5 / 96.1	96.7 / 96.9	95.8 / 96.4	96.9 / 97.0	
	diverse	93.7 / 94.8	96.4 / 96.6	94.1 / 95.6	96.5 / 96.7	diverse	97.0 / 97.1	97.6 / 97.5	97.6 / 97.5	97.7 / 97.5	
	hard+diverse	94.1 / 96.0	96.8 / 96.7	95.1 / 96.9	96.9 / 97.0	hard+diverse	96.4 / 96.9	96.8 / 97.0	96.0 / 96.5	97.2 / 97.2	

Table 12 shows the detection results when using different ID/OOD policy combinations, for library sizes $N = [132, 264, 506, 1012]$, under both environmental conditions. Each entry in the table is formatted as “AUROC-of-detecting-MLRSNet-Hard / AUROC-of-detecting-MLRSNet-Holdout” and is the result of executing the learn-online workflow with the *MixupOE* training algorithm. Note, from Table 11 the baseline numbers to compare these table entries with are 90.4 / 93.9 (from the *MixupOE* (offline) result), which will show whether the learn-online process helped or hurt the detection performance. Finally, we have annotated the table entries, so the **red** numbers indicate the performance of the worst policy choice where the **green** numbers show the performance of the top-performing policy.

From Table 12 we make several interesting observations. First, notice the “- / -” entries for the ID policy of queue in the ordered environment. As suggested in the description of this environment, these exist because after the library correction in 4) there are no ID samples to train on, meaning the 5) model update could not be performed. Next, for a given library size the results in the mixed environment are superior to the results in the ordered environment. This shows that even with the best policy selection, we cannot entirely compensate for the difference in environmental conditions. Finally, in the mixed environments even the worst policy selection leads to performance improvements over the offline model, whereas in the ordered environment a poor policy selection consistently leads to performance degradation.

From these results we can also establish some rules of thumb. In the ordered environment, it is immediately clear that using an OOD policy of *hard+diverse* is the best across all library sizes. In addition, most of the top performing models also use an ID policy of *hard+diverse*. Thus, we conclude that if an ordered environment is anticipated,

both the ID and OOD libraries should employ a *hard+diverse* collection policy. In many ways this follows our intuition. While the *queue* and *diverse* policies are heavily dependent on the sequence in which the data is observed (because old data is evicted in favor of newer data), policies that include an element of hardness ensure that old data can stay in the library if among the most difficult. Then, using hardness and diversity together grants that not all the hardest samples are from the same category.

When anticipating a mixed environment, it is also clear that using an OOD policy of *hard+diverse* is reliably preferred. However, differently from the ordered environment, using a *queue* ID policy consistently leads to the best results. Thus, we conclude that if a mixed environment is expected, the ID policy should be *queue* while the OOD policy should be *hard+diverse*. This signifies that when the environment naturally produces a diverse sampling of data in time, it is less critical that the ID policy enforces hardness or diversity.

5.0 CONCLUSIONS

In this project, we conducted a diverse array of research projects studying the robustness and vulnerability of DNN models in a variety of application settings. We broke our research into two main threads. First, we developed powerful new adversarial attack algorithms which will not only serve as difficult benchmarks for future research, but also allow us to better understand why current models have the vulnerabilities that they do. Next, we devise several approaches for increasing the robustness of DNNs in two important settings: distributed learning scenarios and open-world target recognition environments. Each of these works has been peer reviewed and published in respected global research forums and journals. For additional information about these projects, see their corresponding papers outlined in **Appendix A**.

6.0 REFERENCES

- [1] Goodfellow, I. J., Shlens, J., Szegedy, C., “Explaining and Harnessing Adversarial Examples,” *ICLR 2015*, San Diego California (2015).
- [2] Xiao, H., C. Eckert, “Adversarial label flips attack on support vector machines,” *ECCV 2012*, pp. 870–875, (2012).
- [3] Biggio, B., Nelson, B., Laskov, P., “Poisoning attacks against support vector machines,” *ICML 2012*, (2012).
- [4] Yosinski, J., Clune, J., Bengio, Y., Lipson, H., “How transferable are features in deep neural networks?,” *NeurIPS 2014*, (2014).
- [5] Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J., “Boosting Adversarial Attacks with Momentum,” *CVPR 2018*, Salt Lake City Utah (2018).
- [6] Deng J., Dong, W., Socher, R., Li, L., Li, K., Li, F., “Imagenet: A large-scale hierarchical image database,” in *CVPR 2009*, pp. 248–255. (2009).
- [7] Naseer, M., Khan, S.H., Khan, M.H., Khan, F.S., Porikli, F., “Cross-domain transferability of adversarial perturbations,” *NeurIPS 2019*, (2019).
- [8] McMahan, B., Moore, E., Ramage, D., Hampson, S., Aguera y Arcas, B., “Communication efficient learning of deep networks from decentralized data,” *Artificial Intelligence and Statistics*, (2017).
- [9] Melis, L., Song, C., De Cristofaro, E., Shmatikov, V., “Exploiting unintended feature leakage in collaborative learning,” *2019 IEEE Symposium on Security and Privacy (SP)*, (2019).
- [10] Zhu, L., Liu, Z., Han, S., “Deep leakage from gradients”, *NeurIPS 2019*, (2019).
- [11] Li, A., Sun, J., Wang, B., Duan, L., Li, S., Chen, Y., Li, H., “Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets,” *arXiv:2008.03371*, (2020).
- [12] Lewis, B., Scarnati, T., Sudkamp, E., Nehrbass, J., Rosencrantz, S., Zelnio, E., “A SAR dataset for ATR Development: The synthetic and measured paired labeled experiment (SAMPLE),” *Proc. SPIE*, vol. 10987, Art. no. 109870H, (2019).
- [13] Hendrycks, D., Mazeika, M., Dietterich, T.G., “Deep anomaly detection with outlier exposure,” *ICLR 2019*, New Orleans Louisiana, (2019).
- [14] Ross, T., Worrell, S., Velten, V., Mossing, J., Bryant, M., “Standard SAR ATR evaluation experiments using the MSTAR public release data set,” *Proc. SPIE*, vol. 3370, pp. 566–573, (1998).
- [15] Müller, R., Kornblith, S., Hinton, G.E., “When does label smoothing help?,” *Proc. Adv. Neural Inf. Process. Syst.*, pp. 4696–4705, (2019).
- [16] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A., “Towards deep learning models resistant to adversarial attacks,” *ICLR 2018*, Vancouver BC, (2018).

- [17] Wang, Y., Wang, C., Zhang, H., Dong, Y., Wei, S., “A SAR dataset of ship detection for deep learning under complex backgrounds,” *Remote Sens.*, vol. 11, no. 7, Mar. 2019, Art. no. 765, (2019).
- [18] Liang, S., Li, Y., Srikant, R., “Enhancing the reliability of out-of- distribution image detection in neural networks,” *ICLR 2018*, Vancouver BC, (2018).
- [19] Cheng G., Xie, X., Han, J., Guo, L., Xia, G.S., “Remote sensing image scene classification meets deep learning: Challenges, methods, benchmarks, and opportunities,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 37353756, (2020).
- [20] Inkawhich, N, Inkawhich, M.J., Davis, E.K., Majumder, U.K., Tripp, E., Capraro, C., Chen, Y., “Bridging a gap in sar-atr: Training on fully synthetic and testing on measured data,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 29422955, (2021).
- [21] Hein, M., Andriushchenko, M., Bitterwolf, J., “Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem,” *CVPR 2019*, pp. 41-50, (2019).
- [22] Zhang, J., Inkawhich, N., Chen, Y., Li, H., “Fine-grained out of-distribution detection with mixup outlier exposure,” *CoRR*, vol. abs/2106.03917, (2021).
- [23] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R., “Overcoming catastrophic forgetting in neural networks,” *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, (2017).
- [24] Castro, F.M., Marín-Jiménez, M.J., Guil, N., Schmid, C., Alahari, K., “End-to-end incremental learning,” *ECCV 2018*, pp. 241–257, (2018).
- [25] Li, Z., Hoiem, Z., “Learning without forgetting,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, (2018).
- [26] Kurakin, A., Goodfellow, I.J., Bengio, S., “Adversarial machine learning at scale,” *CoRR*, abs/1611.01236, (2016).
- [27] Kurakin, A., Goodfellow, I.J., Bengio, S., Dong, Y., Liao, F., Liang, M., Pang, T., Zhu, J., Hu, X., Xie, C., Wang, J., Zhang, Z., Ren, Z., Yuille, A.L., Huang, S., Zhao, Y., Zhao, Y., Han, Z., Long, J., Berdibekov, Y., Akiba, T., Tokui, S., Abe, M., “Adversarial attacks and defences competition,” *CoRR*, abs/1804.00097, (2018).
- [28] Smilkov, D., Thorat, N., Kim, B., Viégas, F.B., Wattenberg, “Smoothgrad: removing noise by adding noise,” *arXiv*, abs/1706.03825, 2017.
- [29] Bendale, A., Boulton, T.E., “Towards open set deep networks,” *CVPR 2016*, (2016).
- [30] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., “Playing Atari with Deep Reinforcement Learning,” *CoRR* abs/1312.5602, (2013).

- [31] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., “OpenAI Gym,” *arXiv:1606.01540*, (2016).
- [32] Geiping, J., Bauermeister, H., Dröge, H., Moeller, M., “Inverting gradients—how easy is it to break privacy in federated learning?”, *NeurIPS 2020*, (2020).
- [33] McMahan, B., Ramage, D., Talwar, K., Zhang, L., “Learning differentially private recurrent language models,” *ICLR 2018*, (2018).
- [34] Qi, X., Zhu, P., Wang, Y., Zhang, L., Peng, J., Wu, M., Chen, J., Zhao, X., Zang, N., Mathiopoulos, P.T., “Mlrsnet: A multi-label high spatial resolution remote sensing dataset for semantic scene understanding,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 169, pp. 337–350, (2020).
- [35] Christie, G., Fendley, N., Wilson, J., Mukherjee, R., “Functional map of the world,” *CVPR 2018*, (2018).
- [36] Li, H., Dou, X., Tao, C., Wu, Z., Chen, J., Peng, J., Deng, M., Zhao, L., “Rsi-cb: A large-scale remote sensing image classification benchmark using crowdsourced data,” *Sensors*, vol. 20, no. 6, (2020).
- [37] Li, K., Wan, G., Cheng, G., Meng, L., Han, J., “Object detection in optical remote sensing images: A survey and a new benchmark,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 159, pp. 296–307, (2020).
- [38] Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., Torralba, A., “Places: A 10 million image database for scene recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2017).

APPENDIX A – PUBLICATIONS

- 1) Yang, C., Wu, Q. Li, H., Chen, Y., “Generative poisoning attack method against neural networks,” *CoRR* abs/1703.01340, (2017).
- 2) Inkawhich, N., Wen, W., Li, H., Chen, Y., “Feature space perturbations yield more transferable adversarial examples,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2019*, pp. 7066-7074, (2019).
- 3) Inkawhich, N., Liang, K.J., Carin, L., Chen, Y., “Transferable perturbations of deep feature distributions,” *8th International Conference on Learning Representations (ICLR) 2020*, (2020).
- 4) Inkawhich, M., Chen, Y., Li, H., “Snooping attacks on deep reinforcement learning,” *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2020*, pp. 557-565, (2020).
- 5) Inkawhich, N., Liang, K.J., Wang, B., Inkawhich, M., Carin, L., Chen, Y., “Perturbing across the feature hierarchy to improve standard and strict blackbox attack transferability,” *Advances in Neural Processing Systems 33 (NeurIPS) 2020*, (2020).
- 6) Inkawhich, N., Davis, E.K., Inkawhich, M., Majumder, U.K., Chen, Y., “Training SAR-ATR models for reliable operation in open-world environments,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (JSTARS)*, Vol 14, pp. 3954-3966, (2021).
- 7) Inkawhich, N., Liang, K.J., Zhang, J., Yang, H., Li, H., Chen, Y., “Can targeted adversarial examples transfer when source and target models have no label space overlap?”, *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW) 2021*, (2021).
- 8) Sun, J., Li, A., Wang, B., Yang, H., Li, H., Chen, Y., “Soteria: Provable defense against privacy leakage in federated learning from representation perspective,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2021*, pp. 9311-9319, (2021).
- 9) Inkawhich, N., Zhang, J., Davis, E.K., Luley, R., Chen, Y., “Improving out-of-distribution detection by learning from the deployment environment,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (JSTARS)*, Vol 14, No. 8, (2021).

APPENDIX B – ABSTRACT

Deep neural networks have enabled modern computer vision systems to reach new heights of performance on a variety of challenging tasks. Despite the accuracy and efficiency benefits, the highly parameterized non-linear nature of deep networks makes them very difficult to interpret and prone to failure in the presence of an adversary or anomalous data. This vulnerability makes the integration of these models into our real-world systems troubling. This project has two broad threads: (1) we explore the vulnerability of deep neural networks by developing state-of-the-art adversarial attacks, and (2) we improve the robustness of models in challenging operating environments such as in open-world target recognition and federating learning scenarios.

7.0 – LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AA	Activation Attack
AdvOE	Adversarial Outlier Exposure
AFRL	Air Force Research Lab
ATR	Automatic Target Recognition
AUROC	Area Under the Receiver Operating Characteristic
BCE	Binary Cross Entropy
CAD	Computer Aided Design
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
<i>cor</i>	Correlation Coefficient
DIOR	Detection in Optical Remote sensing
DLG	Deep Leakage from Gradients
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
DQN	Deep Q Network
EO	Electro-Optical
ERM	Empirical Risk Minimization
FDA	Feature Distribution Attack
FGM	Fast Gradient Method
FL	Federated Learning
ID	In-Distribution
lblsm	Label Smoothing

MIFGM	Momentum Iterative Fast Gradient Method
MixupOE	Mixup Outlier Exposure
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
MSTAR	Man-portable Surveillance and Target Acquisition Radar
NIPS	Annual Conference on Neural Information Processing Systems
ODIN	An Out-of-Distribution detection method
OE	Outlier Exposure
OOD	Out-of-Distribution
OSR	Open Set Recognition
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RS	Remote Sensing
RSI-CB	remote sensing image classification benchmark
SAMPLE	Synthetic and Measured Paired Labeled Experiment
SAR	Synthetic Aperture Radar
SGM	Skip Gradient Method
Tanh	Hyperbolic Tangent
TNR@95TPR	True Negative Rate @ 95% True Positive Rate
TMIM	Targeted Momentum Iterative Method
Val	Validation
w.r.t.	With respect to