Indicators of Attack Failure: Debugging and Improving Optimization of Adversarial Examples

Anonymous Author(s)
Affiliation
Address
email

Abstract

Evaluating robustness of machine-learning models to adversarial examples is a challenging problem. Many defenses have been shown to provide a false sense of security by causing gradient-based attacks to fail, and they have been broken under more rigorous evaluations. Although guidelines and best practices have been suggested to improve current adversarial robustness evaluations, the lack of automatic testing and debugging tools makes it difficult to apply these recommendations in a systematic manner. In this work, we overcome these limitations by proposing: (i) a set of quantitative *indicators of failure*, which unveil common failures in the optimization of gradient-based attacks; and (ii) specific fixes to overcome such failures, which are applied according to a systematic evaluation protocol. Our extensive experimental analysis shows that our indicators of failure can be used to visualize, debug and improve current adversarial robustness evaluations, thereby providing a first concrete step towards automatizing and systematizing the process of evaluating adversarial robustness of machine-learning models.

1 Introduction

Despite their unprecedented success in many different applications, machine-learning models have been shown to be vulnerable to *adversarial examples* [29, 3], i.e., inputs intentionally crafted to mislead such models at test time. While hundreds of defenses have been proposed to overcome this issue [34, 22, 31, 27], many of them turned out to only spread a false sense of security, as their robustness to adversarial inputs was significantly overestimated. In particular, some of them were evaluated by running existing attacks for too few iterations or using inappropriate attack hyperparameters, while others were causing gradient-based attacks to fail by making gradients uninformative or obfuscating them. Such defenses have been indeed shown to yield much lower robustness by running more rigorous robustness evaluations and adapting attacks to evade them [9, 1].

To prevent the same evaluation mistakes from occurring multiple times and help develop better defenses, several recommendations have been released [10]. However, their application is non-trivial, and they have been mostly neglected; in particular, 13 defenses published after the release of these checklists were found to be ineffective due to the difficulty of optimizing the attacks correctly [30]. Moreover, the proposed checklists can themselves provide a false sense of security [25], as their application is insufficient to ensure evaluation correctness. While recently-proposed evaluation frameworks may provide more reliable evaluations by combining diverse attacks [11, 32], it is still unclear whether the reported results are affected by subtle failures of the underlying attacks.

To address these limitations, we formalize a systematic methodology to detect and fix the issues that hinder the correct evaluation of machine-learning models' robustness against adversarial examples by providing the following contributions: (i) we introduce a unified attack framework that captures the predominant styles of existing gradient-based attack methods, allowing us to identify six main

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

failures that may arise during their optimization (Sect. 2); (ii) we propose six *indicators of attack failures* (IoAF), i.e., metrics and principles that help understand why and when gradient-based attacks fail (Sect. 3); (iii) we empirically validate our methodology by re-evaluating the robustness of seven recently-published defenses, showing how their evaluations could have been improved by monitoring the IoAF values and following our evaluation protocol (Sect. 4; and (iv) we open source our code and data to enable reproducibility of our findings, at `https://github.com/ioaf-todo`.[1] We conclude by discussing related work (Sect. 5), limitations, and future research directions (Sect. 6).

2 Adversarial Robustness: Gradient-based Attacks and Failures

To identify the main categories of attack failure, we introduce here a unifying framework that encompasses many different gradient-based attack algorithms proposed so far for optimizing adversarial examples. We start by pointing out that such attacks solve a multiobjective optimization problem:

$$\min_{\boldsymbol{\delta} \in \Delta} \left(L(\boldsymbol{x} + \boldsymbol{\delta}, y; \boldsymbol{\theta}), \|\boldsymbol{\delta}\|_p \right) , \tag{1}$$

where $\boldsymbol{x} \in [0,1]^d$ is the input sample, $y \in \{1, \dots, c\}$ is either its label (for untargeted attacks) or the label of the target class (for targeted attacks), $\boldsymbol{\delta} \in \Delta$ is the computed perturbation to achieve misclassification within the feasible input domain, and $\boldsymbol{\theta}$ are the parameters of the model. This problem presents an inherent tradeoff: minimizing L amounts to finding an adversarial example with large misclassification confidence and perturbation size, while minimizing $\|\boldsymbol{\delta}\|_p$ penalizes larger perturbations (in the given ℓ_p norm) at the expense of decreasing misclassification confidence. Typically the loss L is defined as the cross-entropy (CE) loss or the logit difference [9].[2] The choice of the loss function (along with the target model) defines the so-called *loss landscape*, i.e., the high-dimensional surface that the optimization algorithm must navigate to find adversarial examples.

Multiobjective problems can be solved by establishing a different tradeoff between the given objectives along the Pareto frontier, by using either soft- or hard-constraint reformulations. For example, Carlini-Wagner (CW) [9] is a soft-constraint attack, which reformulates Eq.(1) as an unconstrained optimization: $\min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\|_p + c \cdot \min(L(\boldsymbol{x} + \boldsymbol{\delta}, y, \boldsymbol{\theta}), -\kappa)$, where the hyperparameters κ and c tune the trade-off between misclassification confidence and perturbation size. Hard-constraint reformulations instead aim to minimize one objective while constraining the other. They include maximum-confidence attacks like Projected Gradient Descent (PGD) [19], which is formulated as $\min_{\boldsymbol{\delta}} L(\boldsymbol{x} + \boldsymbol{\delta}, y; \boldsymbol{\theta})$ s.t. $\|\boldsymbol{\delta}\|_p \leq \epsilon$, and minimum-norm attacks like Brendel-Bethge (BB) [4], the Decoupling Direction-Norm (DDN) attack [26], and the Fast Minimum-Norm (FMN) attack [23] which can be formulated as $\min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\|_p$ s.t. $L(\boldsymbol{x} + \boldsymbol{\delta}, y; \boldsymbol{\theta}) \leq k$. In these cases, ϵ and k upper bound the perturbation size and the misclassification confidence, respectively, thereby optimizing a different tradeoff between these two. Each of the above attacks often needs to use an approximation $\hat{\boldsymbol{\theta}}$ of the target model, since the original model may be either non-differentiable or not sufficiently smooth [1], hindering the gradient-based attack optimization process. In this case, once the attacker loss has been optimized on the surrogate model $\hat{\boldsymbol{\theta}}$, the attack is considered successful if it evades the target model $\boldsymbol{\theta}$.

Generalized Attack Algorithm. While each of the attacks discussed above minimizes different objectives with different constraints, we observe that they can all be seen as solutions to a common multiobjective problem, based on gradient descent. We summarize their main steps in Algorithm 1. The algorithm starts by define an *initialization point* (line 1), which can be achieved by directly using the input point $\boldsymbol{x}$, a randomly-perturbed version of it, or even a sample from the target class [4]. Then, if the target model $\boldsymbol{\theta}$ is difficult to directly exploit (e.g., because it is non-differentiable) the attacker has to use a surrogate model $\hat{\boldsymbol{\theta}}$ to approximate the target $\boldsymbol{\theta}$ (line 2). The attack then iteratively updates the initial point searching for a better adversarial example (line 4), computing in each iteration one (or more) gradient descent steps (line 5) using the initial point and the perturbation $\boldsymbol{\delta}_i$ computed so far. The new perturbation $\boldsymbol{\delta}_{i+1}$ is obtained by enforcing the constraints defined in the problem (line 6), which can then be updated according to the chosen strategy [23, 26]. For *maximum-confidence* attacks, the perturbation is prevented from exiting the Δ region, by projecting the perturbed input sample onto the feasible domain at each iteration. For *minimum-norm* attacks, instead, success is achieved only if an adversarial example is found inside the Δ region, while fulfilling the misclassification loss constraint. During gradient descent, the attacker collects all the perturbations along the iterations to

[1] Anonymized for submission.

[2] The sign of L can be adjusted internally to properly account for both untargeted and targeted attacks.

Algorithm 1: Generalized gradient-based attack for optimizing adversarial examples.

Input : $\boldsymbol{x}$, the initial point; y, the true class of the initial point; n, the number of iterations; α, the learning rate; f, the target model; Δ, the considered region.

Output : $\boldsymbol{x}^\star$, the solution found by the algorithm

1 $\boldsymbol{x}_0 \leftarrow \texttt{initialize}(\boldsymbol{x})$ ▷ `Initialize starting point`

2 $\hat{\boldsymbol{\theta}} \leftarrow \texttt{approximation}(\boldsymbol{\theta})$ ▷ `Approximate model parameters`

3 $\boldsymbol{\delta}_0 \leftarrow \mathbf{0}$ ▷ `Initial` δ

4 **for** $i \in [1, n]$ **do**

5 $\quad \boldsymbol{\delta}' \leftarrow \boldsymbol{\delta}_i - \alpha \nabla_{\boldsymbol{x}_i} L(\boldsymbol{x}_0 + \boldsymbol{\delta}_i, y; \hat{\boldsymbol{\theta}})$ ▷ `Compute optimizer step`

6 $\quad \boldsymbol{\delta}_{i+1} \leftarrow \texttt{apply-constraints}(\boldsymbol{x}_0, \boldsymbol{\delta}', \Delta)$ ▷ `Apply constraints (if needed)`

7 $\boldsymbol{\delta}^\star \leftarrow \texttt{best}(\boldsymbol{\delta}_0, ..., \boldsymbol{\delta}_n)$ ▷ `Choose best perturbation`

8 **return** $\boldsymbol{\delta}^\star$

form an *attack path*. The algorithm should then return the best perturbation found anywhere in the attack path, i.e., the perturbation for which the objective function is minimized (line 7).

2.1 Gradient-based Attack Failures

We now describe the failures that hinder the success of gradient-based attacks. We identify two main categories of failures, which we refer to respectively as *loss landscape* and *attack optimization* failures. We provide a graphical representation of each failure in Fig. 1.

Loss Landscape Failures. The first category of failures is connected to the definition of the optimization problem given in Eq. (1) and, in particular, to the combination of the loss function and the target model. In principle, these failures can be identified before running any gradient-based attack, by inspecting the values of the loss function around some input points. This category encompasses all failures related to the presence of *obfuscated gradients* [9, 1], which prevents gradient-based attacks to find adversarial examples even when they exist within the feasible domain. We refer to such failures as F_1 and F_2 in Fig. 1, corresponding to *shattered* and *stochastic* gradients, respectively.

***F₁*: Shattered Gradients.** Useful gradients of the model can not be computed, due to the presence of non-differentiable operations, or numerically unstable defense components [22]. This prevents any optimization algorithm from returning useful information, preventing the attack from reaching its objective (F_1 in Fig. 1).

***F₂*: Stochastic Gradients.** Gradients of the model can be computed, but their value is uninformative, as following the gradient direction does not correctly minimize the objective function of the attack. Such issue arises when models introduce randomness inside their inference process [17], or when their training induces noisy gradients [30, 7] (F_2 in Fig. 1).

Attack Optimization Failures. The second category of failures is connected to problems encountered when running a gradient-based attack to solve the optimization problem given in Eq. (1) with Algorithm 1. This category encompasses failures F_3-F_6 in Fig. 1.

***F₃*: Implementation Errors.** If the attack algorithm is unable to find adversarial examples, the implementation might be flawed. For example, we isolated a bug inside the PGD attack procedure proposed by Madry *et al.* [19]. This attack returns the *last* point of the attack path (line 7 of Algorithm 2) even if this point is non-adversarial, discarding potentially successful adversarial examples found earlier along the optimization process (F_3 in Fig. 1). Similarly, a recent work showed how AutoAttack [11] returns the original point when no adversarial examples are found. While this is not a problem *per se*, it becomes faulty when applied against a surrogate model and later transferred to the real target, since it will use the unperturbed point to calculate the adversarial robustness [13].

***F₄*: Non-converging attack.** Once executed, attacks may not reach convergence (F_4 in Fig. 1). This problem can be caused by sub-optimal choices of the attack hyperparameters, including the step size α and the number of iterations n (as shown in Algorithm 1). If α is too small, the gradient update step is not sufficiently exploring the space (line 5 of Algorithm 1), while using too few iterations n might cause an early stopping of the attack (line 4 of Algorithm 1). An example of this failure can be found in the evaluations of the defenses proposed by Buckman *et al.* [5] and Pang *et al.* [21], where they respectively used only 7 and 10 steps of PGD for testing the robustness of their defense.

***F₅*: Non-adaptive attack.** The loss function that the attacker optimizes does not match the actual loss

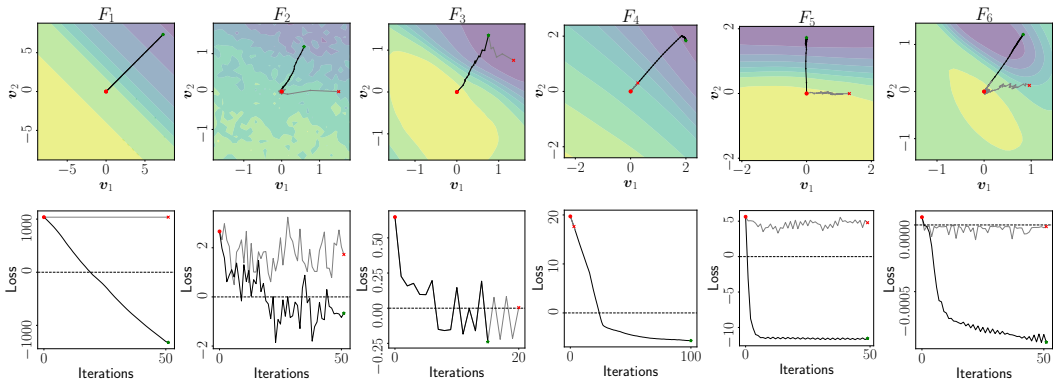

Figure 1: The six attack failures that can be encountered during the optimization of an attack. The failed attack path is shown in *gray*, while the successful attack is displayed in *black*. The point $\boldsymbol{x}_0$ is marked with the *red* dot, the returned point of the failed attack with a *red* cross, and the successful adversarial point with the *green* star. The top row shows the loss landscape, as $L(\boldsymbol{x}+a\boldsymbol{v}_1+b\boldsymbol{v}_2, y_i; \boldsymbol{\theta})$. $\boldsymbol{v}_1$ is the normalized direction $(\boldsymbol{x}_n - \boldsymbol{x}_0)$, while $\boldsymbol{v}_2$ is a representative direction for the displayed case. In the second row we show the value of $L(\boldsymbol{x} + \boldsymbol{\delta}_i, y_i; \boldsymbol{\theta})$ for the evaluated model.

of the target system (*F_5* in Fig. 1). This failure might have different causes. One of them depends on a bad choice of the surrogate model (line 2 of Algorithm 1), and it manifests when either the attack is (i) computed on a undefended model or a portion of the real target, and later tested against the defense; or (ii) the target model is not differentiable, and the chosen surrogate does not correctly approximate it. Another cause lays in the choice of an attack with poor transferability performance, that are later tested on the real target [15] with no success. Since we consider all these cases, we differ from the literature where the term *non-adaptive* has been used only for attacks that were not specifically designed to target a given defense [30]. An example of this failure is found in the evaluation of Das *et al.* [14], where they attack an undefended model, and then evaluated the robustness by directly applying these adversarial examples on the defended model.

***F_6*: Unreachable Misclassification.** The attack runs, it does not encounter any other failure, but it converges to a local minimum without finding any adversarial misclassification (*F_6* in Fig. 1). Since we already have excluded the presence of gradient obfuscation, the causes of this failure are hidden in the combination of both the objective function and the optimizer used to reach the goal, that is not successfully explore the loss landscape of the target model.

3 Indicators of Attack Failure

We now introduce the Indicators of Attack Failure (IoAF): metrics that quantify the presence of failures in the robustness evaluations of machine-learning models. For each indicator, we also identify suitable fixes that can be applied when failures are detected, as summarized in Fig. 2.

***I_1*: Unavailable Gradients.** This indicator detects the presence of non-differentiable components or numerical instabilities when computing gradients (*F_1*). The indicator computes the gradient of the loss of the attack w.r.t. a batch of 10 input samples, triggering if at least one of them has zero norm or if the gradient computation returns an error.

***I_2*: Unstable Predictions.** This indicator quantifies the high variability of the loss landscape, caused by the the presence of noisy gradients (*F_2*). We hence measure how much the CE loss of the model's predictions varies in absolute value when sampled from an ℓ_2-ball with a radius comparable to the step size, centered on an input sample. Let L_0 be the loss at the center of this ball, and $L_1 ... L_n$ the sampled values, the indicator is computed as $\min(\frac{1}{n}\sum_{i=1}^{n} |(L_0 - L_i)/L_0|, 1)$. We repeat this computation on a batch of 10 samples, and the indicator signals the failure when the mean fluctuation exceeds the 10% of the original value. Such a threshold has been decided by observing the empirical results of our work, and that value already signals the presence of the failure.

***I_3*: Silent Success.** This indicator is designed as a binary flag that triggers when the final point in the path is non-adversarial, but a legitimate adversarial example is found inside the attack path, as described by the implementation errors failure (*F_3*).

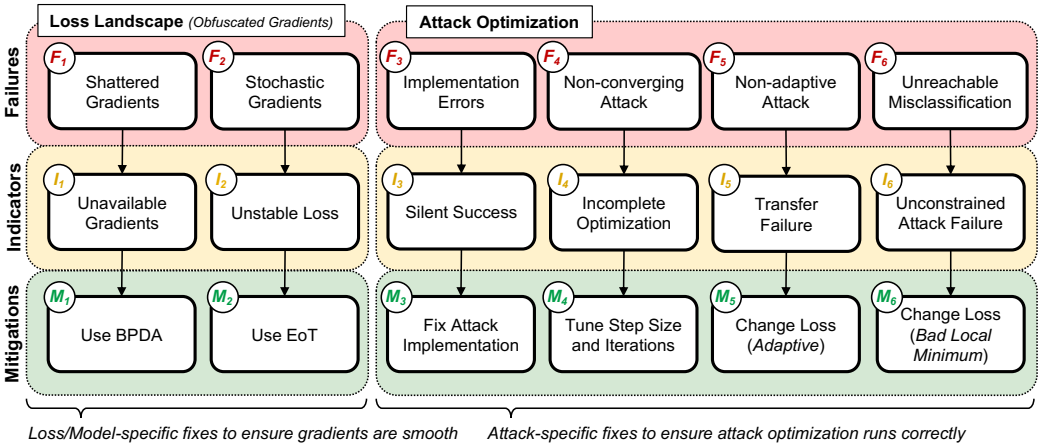

Figure 2: Indicators of Attack Failures. The top row lists the six general failures encountered in gradient-based attacks. The second row lists the Indicators of Attack failures we propose, and the last row depicts possible mitigations that can be applied.

***I_4*: Incomplete Optimization.** This indicator quantifies the non-convergence of the attack (F_4) caused by flawed choices for the number of iterations and the step size. Intuitively, a well-converged attack is characterized by negligible improvement of the objective in the last iterations. This indicator builds a monotonically decreasing curve, by keeping track of the best minimum found at each iteration of the attack, as pictured in Fig. 3. Then, the indicator computes the decrement of the loss of the last 10 iterations, and it returns 1 if it does not decrease by at least 1%. Again, this threshold has been decided by observing the results of our experimental analysis.

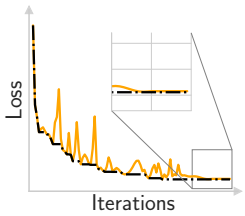

Figure 3: I_4 indicator.

***I_5*: Transfer Failure.** This indicator looks for the non-adaptive failure (F_5) by measuring if the attack fails against the target model while succeeding against the surrogate one. If the attack transfers successfully, the indicator is set to 0, otherwise it is set to 1.

***I_6*: Unconstrained Attack Failure.** This indicator demonstrates the inability of the optimizer to find a misclassification even when using an unbounded perturbation (F_6). This configuration should always lead to successful attacks as the optimization algorithm has no constraints [10]. To measure such behavior, we re-compute adversarial attacks on a set of 10 points by setting an infinite perturbation bound. Hence, this indicator returns 1 if at least one of these attacks fails.

3.1 Fixing Failures

We now propose a systematic evaluation protocol that leverages the output of our indicators of attack failure to improve the reliability of robustness evaluations.

Mitigating Loss Landscape Failures. The evaluation should ensure that the objective function is smooth enough to be explored by the attack algorithm. Hence, the loss landscape failures should be mitigated first, to improve the likelihood of the attack succeeding.

***M_1*: Use BPDA.** If I_1 is active, one of the functions composed to compute the model's predictions $f = f_K \circ \ldots \circ f_i \circ \ldots \circ f_1$ is affecting the computation of the input gradient $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{\partial f_K}{\partial f_{K-1}} \circ \ldots \circ \frac{\partial f_i}{\partial f_{i-1}} \circ \ldots \circ \frac{\partial f_1}{\partial x}$, as it may either be non-differentiable, or cause numerical instabilities. In this case, the Backward Pass Differentiable Approximation (BPDA) [2] can be used to approximate the derivative of the problematic function f_i as $\frac{\partial f_i}{\partial f_{i-1}} = \mathbf{I}$, being $\mathbf{I}$ the identity matrix.

***M_2*: Use EoT.** If I_2 is active, the computed gradients point towards uninformative, random directions that cause the attack to fail. This issue is fixed by applying the *Expectation over Transformation* (EoT) technique [1, 30], which smooths the attack loss L as $\hat{L} = \frac{1}{\gamma} \sum_{i=1}^{\gamma} L(\boldsymbol{x} + \boldsymbol{\delta}_i, y; \boldsymbol{\theta})$, where $\boldsymbol{\delta}_i$ is sampled from an isotropic Normal distribution $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$. The parameters σ and γ have to be chosen based on the problem at hand, to provide sufficiently-informative gradients.

Mitigating Attack Optimization Failures. Once the issues related to gradient obfuscation are solved, we can focus on fixing the failures encountered during the attack optimization.

***M_3*: Fix Attack Implementation.** If I_3 is active, adversarial examples are correctly computed but they are not exploitable since the implementation is flawed. Hence, such attack implementation must be fixed accordingly, by ensuring that the optimization algorithm returns the best result.

***M_4*: Tune Step Size and Iterations.** If I_4 is active, it means that the optimization result can still be improved, and hence both the step size and the number of iterations can be increased.

***M_5*: Change Loss (Adaptive).** If I_5 is active, the attack should be repeated upon changing the surrogate to better approximate the target, choosing a stronger attack, or including the defense inside the attack objective itself [30]. This step implies repeating the previous debugging steps and thus ensuring that the new attack is not suffering from the previous failures.

***M_6*: Change Loss (Bad Local Minimum).** If I_6 is active, the chosen optimization algorithm is not exploring sufficiently the loss landscape of the model. Hence, the evaluation should be repeated with a better attack strategy that converges to adversarial examples when the hard constraints are removed, like applying the normalization of the gradient norm [28, 33].

Let us finally remark that, when no indicator triggers, one should not conclude that robustness may not be worsened by a different, more powerful attack. This is indeed a problem of any empirical evaluation, including software testing – which does not ensure that software is bug-free, but only that some known issues get fixed. Our methodology, similarly, highlights the presence of *known* failures in adversarial robustness evaluations, and suggests how to mitigate them with a systematic protocol, taking a first concrete step towards making robustness evaluations more systematic and reliable.

4 Experiments

We now demonstrate the efficacy of our indicators, by analyzing the robustness evaluations of selected models through the lens of our methodology.

Experimental setup. We run our attacks on an Intel® Xeon® CPU E5-2670 v3, with 48 cores, 126 GB of RAM, equipped with a Nvidia Quadro M6000 with 24 GB of memory, and we leverage the SecML library [24] to implement our methodology. We show failures of gradient-based attacks on 7 previously published defenses [14, 17, 22, 31, 21, 33, 28, 16, 35], by inspecting them with our evaluation protocol. Most of these models [14, 17, 22, 31, 21, 33] are characterized by wrong robustness evaluations, hindered by failures that spread a false sense of security among the community.We choose these defenses in particular because it is known that the original evaluation is incorrect [1, 30]; our goal here is to show that our framework would have been able to identify these errors in the first place even before performing a correct attack. Additionally, we show how our framework could have been used to fix the original attack evaluation for the encountered issues, offering a more accurate robustness evaluation. We also include the evaluation of an adversarial detector [28] to highlight that our framework is able to detect possible failures of this family of classifiers. Lastly, we include the robustness evaluations of a Wide-ResNet [35] and an adversarially-trained model [16], to highlight that our indicators do not trigger when they inspect sound evaluations. We defer details of the selected original evaluations to the Appendix. To demonstrate that automated attacks do not provide guarantees on performing correct evaluations (without human intervention), we also apply the AutoPGD attack [11] with two different loss functions, namely the Cross-Entropy loss (APGD_{CE}) and the Difference of Logits Ratio (APGD_{DLR}). Robustness is computed with the *robust accuracy* (RA) metric, quantified as the ratio of samples classified correctly within a given perturbation bound ϵ. All the robustness evaluations are performed on a pool of 100 samples from the test dataset of the considered model, and for each attack we evaluate the robust accuracy with $\epsilon = 8/255$ for CIFAR models, and $\epsilon = 0.5$ for the MNIST ones.

Identifying and Fixing Attack Failures. We now delve into the description of the considered evaluations, which failures we detect and how we patch them, reporting the results in Table 1.

Correct Evaluations (ST, ADV-T). We first evaluate the robustness analysis of the Wide-ResNet and the adversarially-trained model, by applying PGD with CE loss, with $n = 100$ (number of steps) and $\alpha = 0.03$ (step size). Since no gradient obfuscation techniques have been used, the loss landscape indicators do not trigger. Also, since the attacks smoothly converge, these evaluations do not trigger any attack optimization indicator. Their robust accuracy is, respectively, 0% and 48%.

Table 1: Indicator values (*cols.*) computed on the selected models (*rows*) using different attacks. The robust accuracy (RA) is reported in the last column (best values in bold). The symbol "Tr" denotes a transfer attack. The ✓ represents the detection of a specific failure. We report in parentheses the mean indicator value for I_2, and the fraction of samples which triggered the indicator for I_3, I_4, and I_6.

Model	**Attack**	I_1	I_2	I_3	I_4	I_5	I_6	**RA**
ST	**PGD**							**0.00**
ADV-T	**PGD**							**0.48**
DIST	**Original**	✓					✓(10/10)	0.95
	APGD$_{\text{CE}}$	✓					✓(10/10)	0.99
	APGD$_{\text{DLR}}$							**0.00**
	Patched							**0.01**
k-WTA	**Original**		✓(.41)	✓(23%)	✓(11%)		✓(4/10)	0.67
	APGD$_{\text{CE}}$		✓(.42)		✓(21%)		✓(2/10)	0.35
	APGD$_{\text{DLR}}$		✓(.33)				✓(4/10)	0.28
	Patched				✓(6%)		✓(2/10)	**0.09**
IT	**Original**		✓(>1)	✓(33%)	✓(90%)			0.32
	APGD$_{\text{CE}}$		✓(>1)	✓(3%)			✓(2/10)	0.12
	APGD$_{\text{DLR}}$		✓(>1)	✓(5%)			✓(1/10)	0.12
	Patched				✓(3%)			**0.00**
EN-DV	**Original**				✓(100%)		✓(3/10)	0.48
	APGD$_{\text{CE}}$							**0.00**
	APGD$_{\text{DLR}}$							**0.00**
	Patched							**0.00**
TWS	**Original**				✓(74%)	✓(6%)	✓(8/10)	0.77
	APGD$_{\text{CE}}$							**0.03**
	APGD$_{\text{DLR}}$						✓(7/10)	0.68
	Patched				✓(1%)			**0.01**
JPEG-C	**Original**	✓						0.85
	Original (Tr)					✓(78%)		0.74
	APGD$_{\text{CE}}$ (Tr)					✓(62%)		0.51
	APGD$_{\text{DLR}}$ (Tr)					✓(85%)		0.70
	Patched						✓(1/10)	**0.46**
DNR	**Original**				✓(100%)		✓(10/10)	0.75
	APGD$_{\text{CE}}$				✓(3%)			**0.02**
	APGD$_{\text{DLR}}$						✓(2/10)	0.20
	Patched				✓(1%)			**0.03**

Defensive Distillation (DIST). Papernot *et al.* [22] use *distillation* to train a classifier to saturate the last layer of the network, making the computations of gradients numerically unstable and impossible to calculate, triggering the I_1 indicator, but also I_6 indicator as the optimizer can not explore the space. To patch this evaluation, we apply M_1 to overcome the numerical instability caused by DIST, forcing the attack to leverage the logits of the model instead of computing the softmax. Such a fix reduces the robust accuracy from 95% to 1%. In contrast, APGD$_{\text{DLR}}$ manages to decrease the robust accuracy to 0%, as it avoids the saturation issue by considering only the logits of the model, while APGD$_{\text{CE}}$ triggers the same issues of the original evaluation.

k-Winners Take All (k-WTA). Xiao *et al.* [31] develop a classifier with a very noisy loss landscape, that destroys the meaningfulness of the directions of computed gradients. The original evaluation triggers I_2, since the loss landscape is characterized by frequent fluctuations, but also I_3, as the applied PGD [19] returns the last point of the attack discarding adversarial examples found within the path. Moreover, due to the noise, the attack is not always reaching convergence, as signaled by I_4, also performing little exploration of the space, as flagged by I_6. Hence, we apply M_2, performing the EoT on the loss of PGD [30] with $\gamma = 2000$ and $\sigma = 8/255$, we fix the implementation with M_3, and

increase the iterations with M_4, thus reducing the robust accuracy from 67% to 9%. However, our evaluation still triggers I_4 and I_6, implying that for some points the result can be improved further by increasing the number of steps or the smoothing parameter γ. Both APGD_{CE} and APGD_{DLR} trigger I_2, since they are not applying EoT, and they partially activate I_6, similarly to the original evaluation. Interestingly, APGD_{DLR} converges better than APGD_{CE}, as the latter triggers I_4. Both attacks decrease the robust accuracy to 35% and 28%, a worst estimate than the one computed with the patched attack that directly handles the presence of the gradient obfuscation.

Input Transformations (IT). Guo *et al.* [17] apply random affine *input transformations* to input images, producing a noisy loss landscape that varies at each prediction, thus its original robustness evaluation I_2, I_3, and I_4 indicators, for the same reasons of k-WTA. Again, we apply M_2, M_3 and M_4, using EoT with $\gamma = 200$ and $\sigma = 8/255$, decreasing the robust accuracy from 32% to 0%. Still, for some points (even if adversarial) the objective could still be improved, as I_4 is active for them. Both APGD_{CE} and APGD_{DLR} are able to slightly decrease the robust accuracy to 12%, but they both trigger the I_2 and I_6 indicator since they are not addressing the high variability of the loss landscape.

Ensemble Diversity (EN-DV). Pang *et al.* [21] propose an ensemble model, evaluated with only 10 steps of PGD, thus triggers the I_5 indicator since the attack is not reaching convergence. As a consequence, also the unbounded attack is failing, thus triggering I_6. We apply the M_4 mitigation and we set $n = 100$, as already done by previous work [30]. This fix changes the robust accuracy from 48% to 0%. Not surprisingly, both APGD_{CE} and APGD_{DLR} are able to bring the robust accuracy to 0%, since they both automatically modulate the step size.

Turning a Weakness into a Strength (TWS). Yu *et al.* [33] propose an adversarial example defense, composed of different detectors. For simplicity, we only consider one of these, and we repeat their original evaluation with PGD without the normalization operation on the computed gradients, and $n = 50$. This attack does not converge, triggering I_4, and if fails to navigate the loss landscape due to the very small gradients, triggering also I_6. We apply M_4, by settint $n = 100$, and we use the original implementation of PGD (with the normalization step), hence applying also M_6, achieving a drop in the robust accuracy from 77% to 1%. For only one point, our evaluation triggers I_4, as better convergence could be achieved. Moving forward, APGD_{CE} finds a robust accuracy of 3%, as it is modulating the step size along with using the normalization of the gradient. On the other hand, APGD_{DLR} triggers the I_6 indicator and the robust accuracy remains at 68%, as the attack is not able to avoid the rejection applied by the model.

JPEG Compression (JPEG-C). Das *et al.* [14] pre-process all input samples using *JPEG compression*, before feeding the input to the undefended network. The authors firstly state that they can not directly attack the defense due to its non-differentiability (thus triggering I_1), then they apply the DeepFool [20] attack against the undefended model. This attack has low transferability, as it is not maximizing the misclassification confidence [15]. The robustness evaluation triggers I_5 since the attack computed on the undefended model is not successful on the real defense. To patch this evaluation, we apply M_1, approximating the back-propagation of the JPEG compression layer with BPDA [30], and we apply M_5 by using PGD to maximize the misclassification confidence of the model. This fix is sufficient to reduce its robust accuracy from 74% to 46%. However, I_6 is active, implying that the chosen BPDA strategy might be refined to get a better result. Since APGD can not attack non-differentiable models, we run the attack against the undefended target and we transfer the adversarial examples on the defense. However, both of them trigger I_5 as transfer attacks are not effective, keeping the robust accuracy to 51% and 70%.

Deep Neural Rejection (DNR). Sotgiu *et al.* [28] propose an adversarial detector encoding it as an additional class that captures the presence of attacks. Similarly to TWS, it was evaluated with PGD with no normalization of the norms of gradients. The attack does not converge, as it triggers I_4, but also sometimes gets trapped inside the rejection class, triggering I_6. Hence, we apply both M_4 and M_6, by increasing the number of iterations and considering an attack that avoids minimizing the score of the rejection class. These fixes reduce the robust accuracy from 75% to 3%. Interestingly, only APGD_{CE} is able to reduce the robust accuracy to 2%, also noting that it might be still beaten since it is triggering I_4. On the contrary, APGD_{DLR} is reducing the robust accuracy only to 20%, by also triggering I_6. By manually inspecting the outputs of the model, we found that its scores tend to have all the same values, due to the SVM-RBF kernel that assigns low prediction outputs to samples that are outside the distributions of the training samples. This causes the DLR loss to saturate due to the denominator becoming extremely small, making this attack weak against the DNR defense.

5 Related Work

Robustness Evaluations. Prior work focused on analyzing the quality of robustness evaluations of selected already-published defenses [8, 1, 30, 10], by inspecting and correcting their over-emphasized security claims with sound analysis. On top of this, the authors also suggest qualitative guidelines and best practices to avoid repeating the mistakes of past robustness evaluations. However, these can neither specify the reasons why the attacks fail nor be applied automatically on the tested defense. In contrast, our work is inspired by these suggestions, as it offers a systematic methodology to provide better robustness evaluations, also quantifying why and how attack fails.

Benchmarks. Instead of measuring the robustness of individual models as the prior papers do, researchers of the community proposed benchmarks that provide a complete evaluation framework that can be applied to any future defense, with the ultimate goal of understanding which technique is better. Ling *et al.* [18] propose DEEPSEC, a benchmark that tests several attacks against a wide range of defenses. However, this framework was shown to be flawed by several implementation issues and problems in the configuration of the attacks [6]. Croce *et al.* [12] propose RobustBench [12], a benchmark that accepts state-of-the-art models as submissions that are automatically tested with AutoAttack [11]. Also, the same authors recently introduced alerts inside AutoAttack, displaying textual warnings if the strategy encounters specific issues (e.g. randomness and zero gradients). Yao *et al.* [32] propose the Adaptive AutoAttack evaluation protocol, that automatically tests a variety of attacks, by applying different configurations of parameters and objective functions, hence extending the procedure proposed by AutoAttack.

While all these protocols are able to automatically compute security evaluations and rank defenses by their robustness, they are just applying attacks one after the other, hence missing either the real rationale behind attack failures or possible flaws of the applied techniques, spreading again a false sense of security. In contrast, our methodology improves these processes, as we are able to detect and patch faults encountered while computing adversarial attacks, hence improving the evaluation protocol. Here we imagine that our framework could be used to help these tools automatically detect when their evaluations are incomplete so that they could flag potential errors that should be investigated.

6 Contributions, Limitations and Future Work

In this work, we propose the Indicators of Attack Failures (IoAF), quantitative tests that help debug failures of adversarial robustness evaluations. We use them to develop a pipeline that detects and suggests how to mitigate the encountered failures, improving the reliability of adversarial robustness evaluations. To demonstrate that our indicators are useful, we select 7 defenses that have been previously shown to be weak against adversarial attacks, and we assess the reliability of their *original* evaluations using our indicators. We find that our methodology would have identified errors in *all* of these broken defenses, and could have predicted that future attacks would break them.

An inherent limitation of our methodology is that, similarly to software testing, it does not ensure that an evaluation is failure-free, but only that none of the known failures we characterize and measure is present. It is indeed not possible to guarantee that a gradient-based attack has found the global optimum of a non-convex optimization problem. This is however a general limitation of empirical evaluations. We nevertheless believe that our set of indicators may also help unveil novel kinds of failure, e.g., if attacks with unbounded perturbation keep failing even after fixes are applied, thereby fostering the development of novel mitigations and indicators that will help extend our framework and improve it in the future.

Another avenue for future work is to further automate our analysis, by systematizing the steps required to make an attack adaptive to the defense. Still, our work provides a set of debugging tools and systematic procedures that aim to significantly reduce the number of manual interventions and the required skills to detect and fix flawed robustness evaluations. We firmly believe that integrating our work in current attack libraries and benchmarks will help avoid factual mistakes in adversarial robustness evaluations which have substantially hindered the development of effective defense mechanisms against adversarial examples thus far.

References

[1] A. Athalye, N. Carlini, and D. A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 274–283. JMLR.org, 2018.

[2] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok. Synthesizing robust adversarial examples. In *ICLR*, 2018.

[3] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, *Machine Learning and Knowledge Discovery in Databases (ECML PKDD), Part III*, volume 8190 of *LNCS*, pages 387–402. Springer Berlin Heidelberg, 2013.

[4] W. Brendel, J. Rauber, M. Kümmerer, I. Ustyuzhaninov, and M. Bethge. Accurate, reliable and fast robustness evaluation, 2019.

[5] J. Buckman, A. Roy, C. Raffel, and I. Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*, 2018.

[6] N. Carlini. A critique of the deepsec platform for security analysis of deep learning models, 2019.

[7] N. Carlini and D. Wagner. Defensive distillation is not robust to adversarial examples, 2016.

[8] N. Carlini and D. A. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In B. M. Thuraisingham, B. Biggio, D. M. Freeman, B. Miller, and A. Sinha, editors, *10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 3–14, New York, NY, USA, 2017. ACM.

[9] N. Carlini and D. A. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, pages 39–57. IEEE Computer Society, 2017.

[10] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin. On evaluating adversarial robustness, 2019.

[11] F. Croce and M. Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020.

[12] F. Croce, M. Andriushchenko, V. Sehwag, N. Flammarion, M. Chiang, P. Mittal, and M. Hein. Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670*, 2020.

[13] F. Croce, S. Gowal, T. Brunner, E. Shelhamer, M. Hein, and T. Cemgil. Evaluating the adversarial robustness of adaptive test-time defenses. *arXiv preprint arXiv:2202.13711*, 2022.

[14] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, S. Li, L. Chen, M. E. Kounavis, and D. H. Chau. Shield: Fast, practical defense and vaccination for deep learning using jpeg compression. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 196–204, 2018.

[15] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX security symposium (USENIX security 19)*, pages 321–338, 2019.

[16] L. Engstrom, A. Ilyas, H. Salman, S. Santurkar, and D. Tsipras. Robustness (python library), 2019. URL `https://github.com/MadryLab/robustness`.

[17] C. Guo, M. Rana, M. Cisse, and L. van der Maaten. Countering adversarial images using input transformations. In *International Conference on Learning Representations*, 2018.

[18] X. Ling, S. Ji, J. Zou, J. Wang, C. Wu, B. Li, and T. Wang. Deepsec: A uniform platform for security analysis of deep learning model. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 673–690, 2019. doi: 10.1109/SP.2019.00023.

[19] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

[20] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, 2016.

[21] T. Pang, K. Xu, C. Du, N. Chen, and J. Zhu. Improving adversarial robustness via promoting ensemble diversity. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4970–4979. PMLR, 09–15 Jun 2019. URL `http://proceedings.mlr.press/v97/pang19a.html`.

[22] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, May 2016. doi: 10.1109/SP.2016.41.

[23] M. Pintor, F. Roli, W. Brendel, and B. Biggio. Fast minimum-norm adversarial attacks through adaptive norm constraints. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 20052–20062. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/a709909b1ea5c2bee24248203b1728a5-Paper.pdf`.

[24] M. Pintor, L. Demetrio, A. Sotgiu, M. Melis, A. Demontis, and B. Biggio. secml: Secure and explainable machine learning in python. *SoftwareX*, 18:101095, 2022. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2022.101095. URL `https://www.sciencedirect.com/science/article/pii/S2352711022000656`.

[25] N. Popovic, D. P. Paudel, T. Probst, and L. Van Gool. Gradient obfuscation checklist test gives a false sense of security.

[26] J. Rony, L. G. Hafemann, L. S. Oliveira, I. B. Ayed, R. Sabourin, and E. Granger. Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses, 2019.

[27] K. Roth, Y. Kilcher, and T. Hofmann. The odds are odd: A statistical test for detecting adversarial examples. In *International Conference on Machine Learning*, pages 5498–5507. PMLR, 2019.

[28] A. Sotgiu, A. Demontis, M. Melis, B. Biggio, G. Fumera, X. Feng, and F. Roli. Deep neural rejection against adversarial examples. *EURASIP Journal on Information Security*, 2020(1): 1–10, 2020.

[29] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL `http://arxiv.org/abs/1312.6199`.

[30] F. Tramer, N. Carlini, W. Brendel, and A. Madry. On adaptive attacks to adversarial example defenses. *Advances in Neural Information Processing Systems*, 33, 2020.

[31] C. Xiao, P. Zhong, and C. Zheng. Resisting adversarial attacks by k-winners-take-all. 2020.

[32] C. Yao, P. Bielik, P. Tsankov, and M. Vechev. Automated discovery of adaptive attacks on adversarial defenses. *Advances in Neural Information Processing Systems*, 34, 2021.

[33] T. Yu, S. Hu, C. Guo, W. Chao, and K. Weinberger. A new defense against adversarial images: Turning a weakness into a strength. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, Oct. 2019.

[34] X. Yuan, P. He, Q. Zhu, and X. Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2805–2824, 2019. doi: 10.1109/TNNLS.2018.2886017.

[35] S. Zagoruyko and N. Komodakis. Wide residual networks. In E. R. H. Richard C. Wilson and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016. ISBN 1-901725-59-6. doi: 10.5244/C.30.87. URL `https://dx.doi.org/10.5244/C.30.87`.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We discuss the limitations in Sect. 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In Sect. 6, we specified that the aim of our work is not to break defenses in an harmful way. Our purpose is only to help researchers improving their security evaluation.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code will be submitted as supplementary material, and the instructions for reproducing the experiments are described in Sect. 4.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We describe the experimental protocol in Sect. 4.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The resourced used for the experiments are listed in Sect. 4.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cited all the existing assets used for the experiments.
   (b) Did you mention the license of the assets? [Yes] We cited the authors of the assets, and we provide the list of external assets along with the code.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provide the code for computing the metrics as supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All the assets we used are publicly available.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]