Carefully Blending Adversarial Training and Purification Improves Adversarial Robustness

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this work, we propose a novel adversarial defence mechanism for image classification – CARSO – blending the paradigms of *adversarial training* and *adversarial purification* in a synergistic robustness-enhancing way. The method builds upon an adversarially-trained classifier, and learns to map its *internal representation* associated with a potentially perturbed input onto a distribution of tentative *clean* reconstructions. Multiple samples from such distribution are classified by the same adversarially-trained model, and an aggregation of its outputs finally constitutes the *robust prediction* of interest. Experimental evaluation by a well-established benchmark of strong adaptive attacks, across different image datasets, shows that CARSO is able to defend itself against adaptive *end-to-end white-box* attacks devised for stochastic defences. Paying a modest *clean* accuracy toll, our method improves by a significant margin the *state-of-the-art* for CIFAR-10, CIFAR-100, and TINYIMAGENET-200 ℓ_∞ robust classification accuracy against AUTOATTACK.

1 Introduction

Vulnerability to adversarial attacks [8, 57] – *i.e.* the presence of inputs, usually crafted on purpose, capable of catastrophically altering the behaviour of high-dimensional models [9] – constitutes a major hurdle towards ensuring the compliance of deep learning systems with the behaviour expected by modellers and users, and their adoption in safety-critical scenarios or tightly-regulated environments. This is particularly true for adversarially-*perturbed* inputs, where a norm-constrained perturbation – often hardly detectable by human inspection [48, 5] – is added to an otherwise legitimate input, with the intention of eliciting an anomalous response [34].

Given the widespread nature of the issue [30], and the serious concerns raised about the safety and reliability of data-learnt models in the lack of an appropriate mitigation [7], adversarial attacks have been extensively studied. Yet, obtaining generally robust machine learning (*ML*) systems remains a longstanding issue, and a major open challenge.

Research in the field has been driven by two opposing, yet complementary, efforts. On the one hand, the study of *failure modes* in existing models and defences, with the goal of understanding their origin and developing stronger attacks with varying degrees of knowledge and control over the target system [57, 21, 44, 60]. On the other hand, the construction of increasingly capable defence mechanisms. Although alternatives have been explored [15, 59, 11, 68], most of the latter is based on adequately leveraging *adversarial training* [21, 42, 58, 49, 23, 31, 54, 62, 18, 47], *i.e.* training a *ML* model on a dataset composed of (or enriched with) adversarially-perturbed inputs associated with their correct, *pre-perturbation* labels. In fact, adversarial training has been the only technique capable of consistently providing an acceptable level of defence [24], while still incrementally improving up to the current *state-of-the-art* [18, 47].

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

Another defensive approach is that of *adversarial purification* [53, 66], where a generative model is used – similarly to denoising – to recover a perturbation-free version of the input before classification is performed. Nonetheless, such attempts have generally fallen short of expectations due to inherent limitations of the generative models used in early attempts [45], or due to decreases in robust accuracy[1] when attacked *end-to-end* [25] – resulting in subpar robustness if the defensive structure is known to the adversary [60]. More recently, the rise of diffusion-based generative models [28] and their use for purification have enabled more successful results of this kind [45, 13] – although at the cost of much longer training and inference times, and a much brittler robustness evaluation [13, 38].

In this work, we design a novel adversarial defence for supervised image classification, dubbed CARSO (*i.e.*, Counter-Adversarial Recall of Synthetic Observations). The approach relies on an adversarially-trained classifier (called hereinafter simply *the classifier*), endowed with a stochastic generative model (called hereinafter *the purifier*). Upon classification of a potentially-perturbed input, the latter learns to generate – from the tensor[2] of (pre)activations registered at neuron level in the former – samples from a distribution of plausible, perturbation-free reconstructions. At inference time, some of these samples are classified by the very same *classifier*, and the original input is robustly labelled by aggregating its many outputs. This method – to the best of our knowledge the first attempt to organically merge the *adversarial training* and *purification* paradigms – avoids the vulnerability pitfalls typical of the mere stacking of a purifier and a classifier [25], while still being able to take advantage of independent incremental improvements to adversarial training or generative modelling.

An empirical assessment[3] of the defence in the ℓ_∞ *white-box* setting is provided, using a *conditional* [56, 64] *variational autoencoder* [32, 50] as the purifier and existing *state-of-the-art* adversarially pre-trained models as classifiers. Such choices are meant to give existing approaches – and the *adversary* attacking our architecture *end-to-end* as part of the assessment – the strongest advantage possible. Yet, in all scenarios considered, CARSO improves significantly the robustness of the pre-trained classifier – even against attacks specifically devised to fool stochastic defences like ours. Remarkably, with a modest *clean* accuracy toll, our method improves by a significant margin the current *state-of-the-art* for CIFAR-10 [33], CIFAR-100 [33], and TINYIMAGENET-200 [14] ℓ_∞ robust classification accuracy against AUTOATTACK [17].

In summary, the paper makes the following contributions:

- The description of CARSO, a novel adversarial defence method synergistically blending *adversarial training* and *adversarial purification*;
- A collection of relevant technical details fundamental to its successful training and use, originally developed for the *purifier* being a *conditional variational autoencoder* – but applicable to more general scenarios as well;
- Experimental assessment of the method, against standardised benchmark adversarial attacks – showing higher robust accuracy *w.r.t.* to existing *state-of-the-art* adversarial training and purification approaches.

The rest of the manuscript is structured as follows. In section 2 we provide an overview of selected contributions in the fields of *adversarial training* and *purification-based* defences – with focus on image classification. In section 3, a deeper analysis is given of two integral parts of our experimental assessment: PGD adversarial training and conditional variational autoencoders. Section 4 is devoted to the intuition behind CARSO, its architectural description, and the relevant technical details that allow it to work. Section 5 contains details about the experimental setup, results, comments, and limitations. Section 6 concludes the paper and outlines directions of future development.

2 Related work

***Adversarial training* as a defence** The idea of training a model on adversarially-generated examples as a way to make it more robust can be traced back to the very beginning of research in the area.

[1] The *test set accuracy* of the frozen-weights trained classifier – computed on a dataset entirely composed of adversarially-perturbed examples generated against that specific model.

[2] Which we call *internal representation*.

[3] Implementation of the method and code for the experiments (based on *PyTorch* [46], `AdverTorch` [19], and `ebtorch` [4]) can be found in the *supplementary materials* of the paper.

The seminal [57] proposes to perform training on a mixed collection of *clean* and adversarial data, generated beforehand.

The introduction of FGSM [21] enables the efficient generation of adversarial examples along the training, with a single normalised gradient step. Its iterative counterpart PGD [42] – discussed in section 3 and Appendix A – significantly improves the effectiveness of adversarial examples produced, making it still the *de facto* standard for the synthesis of adversarial training inputs [24]. Further incremental improvements have also been developed, some focused specifically on robustness assessment (*e.g.* adaptive-stepsize variants, as in [17]).

The most recent adversarial training protocols further rely on synthetic data to increase the numerosity of training datapoints [23, 49, 62, 18, 47], and adopt adjusted loss functions to balance robustness and accuracy [67] or generally foster the learning process [18]. The entire model architecture may also be tuned specifically for the sake of robustness enhancement [47]. At least some of such ingredients are often required to reach the current *state-of-the-art* in robust accuracy via adversarial training.

***Purification* as a defence** Amongst the first attempts of *purification-based* adversarial defence, [25] investigates the use of denoising autoencoders [61] to recover examples free from adversarial perturbations. Despite its effectiveness in the denoising task, the method may indeed *increase* the vulnerability of the system when attacks are generated against it *end-to-end*. The contextually proposed improvement adds a smoothness penalty to the reconstruction loss, partially mitigating such downside [25]. Similar in spirit, [39] tackles the issue by computing the reconstruction loss between the last-layers representations of the frozen-weights attacked classifier, respectively receiving, as input, the *clean* and the tentatively *denoised* example.

In [52], *Generative Adversarial Networks* (GANs) [22] learnt on *clean* data are used at inference time to find a plausible synthetic example – close to the perturbed input – belonging to the unperturbed data manifold. Despite encouraging results, the delicate training process of GANs and the existence of known failure modes [70] limit the applicability of the method. More recently, a similar approach [27] employing *energy-based models* [37] suffered from poor sample quality [45].

Purification approaches based on (conditional) variational autoencoders include [29] and [53]. Very recently, a technique combining variational manifold learning with a test-time iterative purification procedure has also been proposed [65].

Finally, already-mentioned techniques relying on *score-* [66] and *diffusion-* based [45, 13] models have also been developed, with generally favourable results – often balanced in practice by longer training and inference times, and a much more fragile robustness assessment [13, 38].

3 Preliminaries

PGD adversarial training The task of finding model parameters robust to adversarial perturbations is framed by [42] as a *min-max* optimisation problem seeking to minimise *adversarial risk*. The inner optimisation (*i.e.*, the generation of worst-case adversarial examples) is solved by an iterative algorithm – *Projected Gradient Descent* – interleaving gradient ascent steps in input space with the eventual projection on the shell of an ϵ-ball centred around an input datapoint, thus imposing a perturbation strength constraint.

In this manuscript, we will use the shorthand notation ϵ_p to denote ℓ_p norm-bound perturbations of maximum magnitude ϵ.

The formal details of such method are provided in Appendix A.

(Conditional) variational autoencoders Variational autoencoders (*VAE*s) [32, 50] allow the learning from data of approximate generative latent-variable models of the form $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x} \mid \boldsymbol{z})p(\boldsymbol{z})$, whose likelihood and posterior are approximately parameterised by deep artificial neural networks (*ANN*s). The problem is cast as the maximisation of a variational lower bound.

In practice, optimisation is performed iteratively – on a loss function given by the linear mixture of data-reconstruction loss and empirical *KL* divergence *w.r.t.* a chosen prior, computed on mini-batches of data.

Conditional Variational Autoencoders [56, 64] extend *VAE*s by attaching a *conditioning tensor* $\boldsymbol{c}$ – expressing specific characteristics of each example – to both $\boldsymbol{x}$ and $\boldsymbol{z}$ during training. This allows the learning of a decoder model capable of conditional data generation.

Further details on the functioning of such models are given in Appendix B.

4 Structure of CARSO

The core ideas informing the design of our method are driven more by *first principles* rather than arising from specific contingent requirements. This section discusses such ideas, the architectural details of CARSO, and a group of technical aspects fundamental to its training and inference processes.

4.1 Architectural overview and principle of operation

From an architectural point of view, CARSO is essentially composed of two *ANN* models – a *classifier* and a *purifier* – operating in close synergy. The former is trained on a given classification task, whose inputs might be adversarially corrupted at inference time. The latter learns to generate samples from a distribution of potential input reconstructions, tentatively free from adversarial perturbations. Crucially, the *purifier* has only access to the internal representation of the *classifier* – and not even directly to the perturbed input – to perform its task.

During inference, for each input, the internal representation of the *classifier* is used by the *purifier* to synthesise a collection of tentatively unperturbed input reconstructions. Those are classified by the same *classifier*, and the resulting outputs are aggregated into a final *robust prediction*.

There are no specific requirements for the classifier, whose training is completely independent of the use of the model as part of CARSO. However, training it adversarially improves significantly the *clean* accuracy of the overall system, allowing it to benefit from established adversarial training techniques.

The purifier is also independent of specific architectural choices, provided it is capable of stochastic conditional data generation at inference time, with the internal representation of the classifier used as the conditioning set.

In the rest of the paper, we employ a *state-of-the-art* adversarially pre-trained WIDERESNET model as the classifier, and a purpose-built *conditional variational autoencoder* as the purifier, the latter operating decoder-only during inference. Such choice was driven by the deliberate intention to assess the adversarial robustness of our method in its worst-case scenario against a *white-box* attacker, and with the least advantage compared to existing approaches based solely on adversarial training.

In fact, the decoder of a conditional VAE allows for exact algorithmic differentiability [6] *w.r.t.* its conditioning set, thus averting the need for backward-pass approximation [2] in generating *end-to-end* adversarial attacks against the entire system, and preventing (un)intentional robustness by gradient obfuscation [2]. The same cannot be said [13] for more capable and modern purification models, such as those based *e.g.* on diffusive processes, whose robustness assessment is still in the process of being understood [38].

A downside of such choice is represented by the reduced effectiveness of the decoder in the synthesis of complex data, due to well-known model limitations. In fact, we experimentally observe a modest increase in reconstruction cost for non-perturbed inputs, which in turn may limit the *clean* accuracy of the entire system. Nevertheless, we defend the need for a fair and transparent robustness evaluation, such as the one provided by the use of a VAE-based purifier, in the evaluation of any novel architecture-agnostic adversarial defence technique.

A diagram of the whole architecture is shown in Figure 1, and its detailed principles of operation are recapped below.

Training At training time, adversarially-perturbed examples are generated against the *classifier*, and fed to it. The tensors containing the *classifier* (pre)activations across the network are then extracted. Finally, the conditional *VAE* serving as the *purifier* is trained on perturbation-free input reconstruction, conditional on the corresponding previously extracted internal representations, and using pre-perturbation examples as targets.

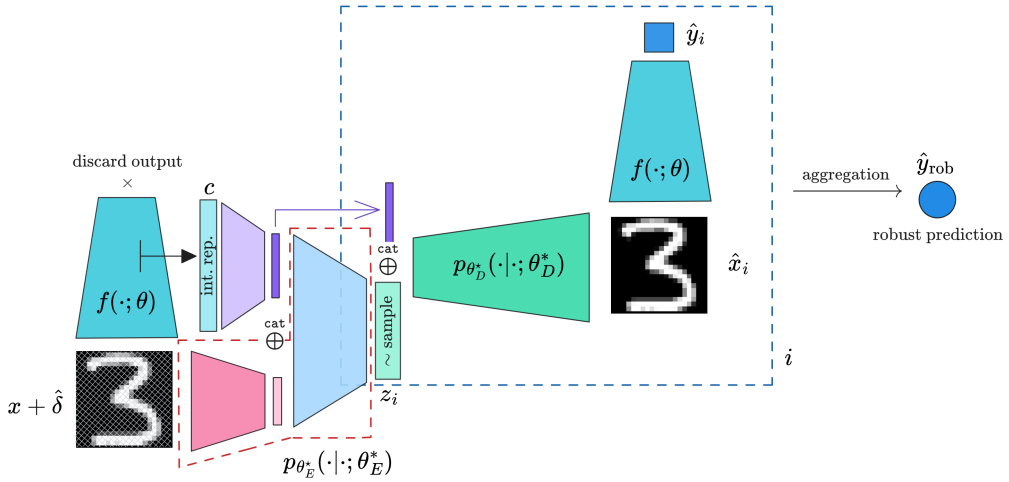

Figure 1: Schematic representation of the CARSO architecture used in the experimental phase of this work. The subnetwork bordered by the red dashed line is used only during the training of the *purifier*. The subnetwork bordered by the blue dashed line is re-evaluated on different random samples $\boldsymbol{z}_i$ and the resulting individual $\hat{y}_i$ are aggregated into $\hat{y}_{\text{rob}}$. The *classifier* $f(\cdot;\boldsymbol{\theta})$ is always kept frozen; the remaining network is trained on $\mathcal{L}_{\text{VAE}}(\boldsymbol{x}, \hat{\boldsymbol{x}})$. More precise details on the functioning of the networks are provided in subsection 4.1.

Upon completion of the training process, the encoder network may be discarded as it will not be used for inference.

Inference The example requiring classification is fed to the *classifier*. Its corresponding internal representation is extracted and used to condition the generative process described by the decoder of the *VAE*. Stochastic latent variables are repeatedly sampled from the original priors, which are given by an *i.i.d.* multivariate Standard Normal distribution. Each element in the resulting set of reconstructed inputs is classified by the same *classifier*, and the individually predicted class logits are aggregated. The result of such aggregation constitutes the robust prediction of the input class.

Remarkably, the only link between the initial potentially-perturbed input and the resulting purified reconstructions (and thus the predicted class) is through the *internal representation* of the classifier, which serves as a *featurisation* of the original input. The whole process is exactly differentiable *end-to-end*, and the only potential hurdle to the generation of adversarial attacks against the entire system is the stochastic nature of the decoding – which is easily tackled by *Expectation over Transformation* [3].

4.2 A first-principles justification

If we consider a trained *ANN* classifier, subject to a successful adversarial attack by means of a slightly perturbed example, we observe that – both in terms of ℓ_p magnitude and human perception – a small variation on the input side of the network is amplified to a significant amount on the output side, thanks to the layerwise processing by the model. Given the deterministic nature of such processing at inference time, we speculate that the *trace* obtained by sequentially collecting the (pre)activation values within the network, along the forward pass, constitutes a richer characterisation of such an amplification process compared to the knowledge of the input alone. Indeed, as we do, it is possible to learn a direct mapping from such featurisation of the input, to a distribution of possible perturbation-free input reconstructions – taking advantage of such characterisation.

4.3 Hierarchical input and internal representation encoding

Training a conditional VAE requires [56] that the conditioning set $\boldsymbol{c}$ is concatenated to the input $\boldsymbol{x}$ before encoding occurs, and to the sample of latent variables $\boldsymbol{z}$ right before decoding. The same is

also true, with the suitable adjustments, for any conditional generative approach where the target and the conditioning set must be processed jointly.

In order to ensure the usability and scalability of CARSO across the widest range of input data and classifier models, we propose to perform such processing in a hierarchical and partially disjoint fashion between the input and the conditioning set. In principle, the encoding of $\boldsymbol{x}$ and $\boldsymbol{c}$ can be performed by two different and independent subnetworks, until some form of joint processing must occur. This allows to retain the overall architectural structure of the purifier, while having finer-grained control over the inductive biases [43] deemed the most suitable for the respective variables.

In the experimental phase of our work, we encode the two variables independently. The input is compressed by a multilayer convolutional neural network (CNN). The internal representation – which in our case is composed of differently sized multi-channel *images* – is processed *layer by layer* by independent multilayer CNNs (responsible for encoding local information), whose flattened outputs are finally concatenated and compressed by a fully-connected layer (modelling inter-layer correlations in the representation). The resulting compressed input and conditioning set are then further concatenated and jointly encoded by a fully-connected network (FCN).

In order to use the VAE decoder at inference time, the entire compression machinery for the conditioning set must be preserved after training, and used to encode the internal representations extracted. The equivalent input encoder may be discarded instead.

4.4 Adversarially-balanced batches

Training the purifier in representation-conditional input reconstruction requires having access to adversarially-perturbed examples generated against the classifier, and to the corresponding clean data. Specifically, we use as input a mixture of *clean* and adversarially *perturbed* examples, and the clean input as the target.

Within each epoch, the *training set* of interest is shuffled [51, 10], and only a fixed fraction of each resulting batch is adversarially perturbed. Calling ϵ the maximum ℓ_p perturbation norm bound for the threat model against which the *classifier* was adversarially pre-trained, the portion of perturbed examples is generated by an even split of $\text{FGSM}_{\epsilon/2}$, $\text{PGD}_{\epsilon/2}$, FGSM_{ϵ}, and PGD_{ϵ} attacks.

Any smaller subset of attack types and strengths, or a detailedly unbalanced batch composition, always experimentally results in a worse performing purification model. More details justifying such choice are provided in Appendix C.

4.5 Robust aggregation strategy

At inference time, many different input reconstructions are classified by the *classifier*, and the respective outputs concur to the settlement of a *robust prediction*.

Calling l_i^α the output logit associated with class $i \in \{1, \dots, C\}$ in the prediction by the classifier on sample $\alpha \in \{1, \dots, N\}$, we adopt the following aggregation strategy:

$$P_i := \frac{1}{Z} \prod_{\alpha=1}^{N} e^{e^{l_i^\alpha}}$$

with P_i being the aggregated probability of membership in class i, Z a normalisation constant such that $\sum_{i=1}^{C} P_i = 1$, and e Euler's number.

Such choice produces a *robust prediction* much harder to take over in the event that an adversary selectively targets a specific input reconstruction. A heuristic justification for this property is given in Appendix D.

5 Experimental assessment

Experimental evaluation of our method is carried out in terms of *robust* and *clean* image classification accuracy within three different scenarios (*a*, *b* and *c*), determined by the specific classification task.

The *white-box* threat model with a fixed ℓ_∞ norm bound is assumed throughout, as it generally constitutes the most demanding setup for adversarial defences.

5.1 Setup

Data The CIFAR-10 [33] dataset is used in *scenario (a)*, the CIFAR-100 [33] dataset is used in *scenario (b)*, whereas the TINYIMAGENET-200 [14] dataset is used in *scenario (c)*.

Architectures A WIDERESNET-28-10 model is used as the *classifier*, adversarially pre-trained on the respective dataset – the only difference between scenarios being the number of output logits: 10 in *scenario (a)*, 100 in *scenario (b)*, and 200 in *scenario (c)*.

The purifier is composed of a conditional VAE, processing inputs and internal representations in a partially disjoint fashion, as explained in subsection 4.3. The input is compressed by a two-layer CNN; the internal representation is instead processed layerwise by independent CNNs (three-layered in *scenarios (a)* and *(b)*, four-layered in *scenario (c)*) whose outputs are then concatenated and compressed by a fully-connected layer. A final two-layer FCN jointly encodes the compressed input and conditioning set, after the concatenation of the two. A six-layer deconvolutional network is used as the decoder.

More precise details on all architectures are given in Appendix E.

Outer minimisation In *scenarios (a)* and *(b)*, the *classifier* is trained according to [18]; in *scenario (c)*, according to [62]. *Classifiers* were always acquired as pre-trained models, using publicly available weights provided by the respective authors.

The *purifier* is trained on the *VAE* loss, using *summed pixel-wise channel-wise* binary cross-entropy as the reconstruction cost. Optimisation is performed by RADAM+LOOKAHEAD [41, 69] with a learning rate schedule that presents a linear warm-up, a plateau phase, and a linear annealing [55]. To promote the learning of meaningful reconstructions during the initial phases of training, the *KL divergence* term in the VAE loss is suppressed for an initial number of epochs. Afterwards, it is linearly modulated up to its actual value, during a fixed number of epochs (β *increase*) [26]. The initial and final epochs of such modulation are reported in Table 14.

Additional scenario-specific details are provided in Appendix E.

Inner minimisation $\epsilon_\infty = {}^8\!/_{255}$ is set as the perturbation norm bound.

Adversarial examples against the *purifier* are obtained, as explained in subsection 4.4, by $\text{FGSM}_{\epsilon/2}$, $\text{PGD}_{\epsilon/2}$, FGSM_{ϵ}, and PGD_{ϵ}, in a *class-untargeted* fashion on the cross-entropy loss. In the case of PGD, gradient ascent with a step size of $\alpha = 0.01$ is used.

The complete details and hyperparameters of the attacks are described in Appendix E.

Evaluation In each scenario, we report the *clean* and *robust* test-set accuracy – the latter by means of AUTOATTACK [17] – of the *classifier* and the corresponding CARSO architecture.

For the *classifier* alone, the *standard* version of AUTOATTACK (*AA*) is used: *i.e.*, the worst-case accuracy on a mixture of AUTOPGD on the cross-entropy loss [17] with 100 steps, AUTOPGD on the *difference of logits ratio* loss [17] with 100 steps, FAB [16] with 100 steps, and the *black-box* SQUARE attack [1] with 5000 queries.

In the evaluation of the CARSO architecture, the number of reconstructed samples per input is set to 8, the logits are aggregated as explained in subsection 4.5, and the output class is finally selected as the $\arg\max$ of the aggregation. Due to the stochastic nature of the *purifier*, robust accuracy is assessed by a version of AUTOATTACK suitable for stochastic defences (*randAA*) – composed of AUTOPGD on the cross-entropy and *difference of logits ratio* losses, across 20 *Expectation over Transformation* (EOT) [3] iterations with 100 gradient ascent steps each.

Computational infrastructure All experiments were performed on an *NVIDIA DGX A100* system. Training in *scenarios (a)* and *(c)* was run on 8 *NVIDIA A100* GPUs with 40 GB of dedicated memory each; in *scenario (b)* 4 of such devices were used. Elapsed real training time for the purifier in all scenarios is reported in Table 1.

Table 1: Elapsed real running time for training the *purifier* in the different scenarios considered.

Scenario	*(a)*	*(b)*	*(c)*
Elapsed real training time	159 min	138 min	213 min

5.2 Results and discussion

An analysis of the experimental results is provided in the subsection that follows, whereas their systematic exposition is given in Table 2.

Table 2: Clean (results in *italic*) and adversarial (results in upright) accuracy for the different models and datasets used in the respective scenarios. The following abbreviations are used: `Scen`: scenario considered; `AT/Cl`: clean accuracy for the adversarially-pretrained model used as the *classifier*, when considered alone; `C/Cl`: clean accuracy for the CARSO architecture; `AT/AA`: robust accuracy (by the means of AUTOATTACK) for the adversarially-pretrained model used as the *classifier*, when considered alone; `C/randAA`: robust accuracy for the CARSO architecture, when attacked *end-to-end* by AUTOATTACK for randomised defences; `Best AT/AA`: best robust accuracy result for the respective dataset (by the means of AUTOATTACK), obtained by adversarial training alone (any model); `Best P/AA`: best robust accuracy result for the respective dataset (by the means of AUTOATTACK), obtained by adversarial purification (any model). Robust accuracies in round brackets are obtained using the PGD+EOT [38] pipeline, developed for diffusion-based purifiers. The best clean and robust accuracies per dataset are shown in **bold**. The clean accuracies for the models referred to in the `Best` columns are shown in Table 15 (in Appendix F).

Scen.	Dataset	AT/Cl	C/Cl	AT/AA	C/rand-AA (PGD+EOT)	Best AT/AA	Best P/AA (PGD+EOT)
(a)	CIFAR-10	***0.9216***	*0.8686*	0.6773	**0.7613** (0.7689)	0.7107	0.7812 (0.6641)
(b)	CIFAR-100	***0.7385***	*0.6806*	0.3918	**0.6665**	0.4267	0.4609
(c)	TINYIMAGENET-200	***0.6519***	*0.5632*	0.3130	**0.5356**	0.3130	

Scenario (a) Comparing the robust accuracy of the *classifier* model used in *scenario (a)* [18] with that resulting from the inclusion of the same model in the CARSO architecture, we observe a $+8.4\%$ increase. This is counterbalanced by a -5.6% clean accuracy toll. The same version of CARSO further provides a $+5.03$ robustness increase *w.r.t.* the current best AT-trained model [47] that employs a $\sim 3\times$ larger RAWIDERESNET-70-16 model.

In addition, our method provides a remarkable $+9.72\%$ increase in robust accuracy *w.r.t.* to the best adversarial purification approach [40], a diffusion-based purifier. However, the comparison is not as straightforward. In fact, the paper [40] reports a robust accuracy of 78.12% using AUTOATTACK on the gradients obtained via the adjoint method [45]. As noted in [38], such evaluation (which uses the version of AUTOATTACK that is unsuitable for stochastic defences) leads to a large overestimation of the robustness of diffusive purifiers. As suggested in [38], the authors of [40] re-evaluate the robust accuracy according to a more suitable pipeline (PGD+EOT, whose hyperparameters are shown in Table 12), obtaining a much lower robust accuracy of 66.41%. Consequently, we repeat the same evaluation for CARSO and compare the worst-case robustness amongst the two. In line with typical AT methods, and unlike diffusive purification, the robustness of CARSO assessed by means of *randAA* is still lower *w.r.t.* than achieved by PGD+EOT.

Scenario (b) Moving to *scenario (b)*, CARSO achieves a robust accuracy increase of $+27.47\%$ *w.r.t.* the *classifier* alone [18], balanced by a 5.79% decrease in clean accuracy. Our approach also improves upon the robust accuracy of the best AT-trained model [62] (WIDERESNET-70-16) by 23.98%. In the absence of a reliable robustness evaluation by means of PGD+EOT for the best purification-based method [40], we still obtain a $+20.25\%$ increase in robust accuracy upon its (largely overestimated) AA result.

Scenario (c) In *scenario (c)*, CARSO improves upon the *classifier* alone [62] (which is also the best AT-based approach for TINYIMAGENET-200) by $+22.26\%$. A significant clean accuracy toll is

imposed by the relative complexity of the dataset, *i.e.* -8.87%. In this setting, we lack any additional purification-based methods.

Assessing the impact of *gradient obfuscation* Although the architecture of CARSO is algorithmically differentiable *end-to-end* – and the integrated diagnostics of the *randAA* routines raised no warnings during the assessment – we additionally guard against the eventual gradient obfuscation [2] induced by our method by repeating the evaluation at $\epsilon_\infty = 0.95$, verifying that the resulting robust accuracy stays below random chance [12]. Results are shown in Table 3.

Table 3: Robust classification accuracy against AUTOATTACK, for $\epsilon_\infty = 0.95$, as a way to assess the (lack of) impact of *gradient obfuscation* on robust accuracy evaluation.

Scenario	*(a)*	*(b)*	*(c)*
$\epsilon_\infty = 0.95$ *acc.*	$<$0.047	$<$0.010	$\approx$0.0

5.3 Limitations and open problems

In line with recent research aiming at the development of robust defences against multiple perturbations [20, 35], our method determines a decrease in *clean* accuracy *w.r.t.* the original model on which it is built upon – especially in *scenario (c)* as the complexity of the dataset increases. This phenomenon is partly dependent on the choice of a VAE as the generative purification model, a requirement for the fairest evaluation possible in terms of robustness.

Yet, the issue remains open: is it possible to devise a CARSO-like architecture capable of the same – if not better – robust behaviour, which is also competitively accurate on clean inputs? Potential avenues for future research may involve the development of CARSO-like architectures in which representation-conditional data generation is obtained by means of diffusion or score-based models. Alternatively, incremental developments aimed at improving the cross-talk between the purifier and the final classifier may be pursued.

Lastly, the scalability of CARSO could be strongly improved by determining whether the internal representation used in conditional data generation may be restricted to a smaller subset of layers, while still maintaining the general robustness of the method.

6 Conclusion

In this work, we presented a novel adversarial defence mechanism tightly integrating input purification, and classification by an adversarially-trained model – in the form of representation-conditional data purification. Our method is able to improve upon the current *state-of-the-art* in CIFAR-10, CIFAR-100, and TINYIMAGENET ℓ_∞ robust classification, *w.r.t.* both *adversarial training* and *purification* approaches alone.

Such results suggest a new synergistic strategy to achieve adversarial robustness in visual tasks and motivate future research on the application of the same design principles to different models and types of data.

References

[1] Maksym Andriushchenko et al. 'Square Attack: a query-efficient black-box adversarial attack via random search'. In: *16th European Conference on Computer Vision*. 2020.

[2] Anish Athalye, Nicholas Carlini and David Wagner. 'Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples'. In: *Proceedings of the International Conference on Machine Learning*. 2018.

[3] Anish Athalye et al. 'Synthesizing Robust Adversarial Examples'. In: *Proceedings of the International Conference on Machine Learning*. 2018.

[4] Emanuele Ballarin. *ebtorch: Collection of PyTorch additions, extensions, utilities, uses and abuses*. 2024. URL: `https://github.com/emaballarin/ebtorch`.

[5] Vincent Ballet et al. 'Imperceptible Adversarial Attacks on Tabular Data'. In: *Thirty-third Conference on Neural Information Processing Systems, Workshop on Robust AI in Financial Services: Data, Fairness, Explainability, Trustworthiness, and Privacy (Robust AI in FS)*. 2019.

[6] Atilim Gunes Baydin et al. 'Automatic differentiation in machine learning: a survey'. In: *The Journal of Machine Learning Research* 18.153 (2018), pp. 1–43.

[7] Battista Biggio and Fabio Roli. 'Wild patterns: Ten years after the rise of adversarial machine learning'. In: *Pattern Recognition* 84 (2018), pp. 317–331.

[8] Battista Biggio et al. 'Evasion Attacks against Machine Learning at Test Time'. In: *Proceedings of the 2013th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part III*. 2013.

[9] Luca Bortolussi and Guido Sanguinetti. *Intrinsic Geometric Vulnerability of High-Dimensional Artificial Intelligence*. 2018. arXiv: `1811.03571`.

[10] Léon Bottou. 'On-Line Algorithms and Stochastic Approximations'. In: *On-Line Learning in Neural Networks*. Cambridge University Press, 1999. Chap. 2.

[11] Ginevra Carbone et al. 'Robustness of Bayesian Neural Networks to Gradient-Based Attacks'. In: *Advances in Neural Information Processing Systems*. 2020.

[12] Nicholas Carlini et al. *On Evaluating Adversarial Robustness*. 2019. arXiv: `1902.06705`.

[13] Huanran Chen et al. *Robust Classification via a Single Diffusion Model*. 2023. arXiv: `2305.15241`.

[14] Patryk Chrabaszcz, Ilya Loshchilov and Frank Hutter. *A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets*. 2017. arXiv: `1707.08819`.

[15] Moustapha Cisse et al. 'Parseval Networks: Improving Robustness to Adversarial Examples'. In: *Proceedings of the International Conference on Machine Learning*. 2017.

[16] Francesco Croce and Matthias Hein. *Minimally distorted Adversarial Examples with a Fast Adaptive Boundary Attack*. 2020. arXiv: `1907.02044`.

[17] Francesco Croce and Matthias Hein. 'Reliable Evaluation of Adversarial Robustness with an Ensemble of Diverse Parameter-free Attacks'. In: *Proceedings of the International Conference on Machine Learning*. 2020.

[18] Jiequan Cui et al. *Decoupled Kullback-Leibler Divergence Loss*. 2023. arXiv: `2305.13948`.

[19] Gavin Weiguang Ding, Luyu Wang and Xiaomeng Jin. *AdverTorch v0.1: An Adversarial Robustness Toolbox based on PyTorch*. 2019. arXiv: `1902.07623`.

[20] Hadi M. Dolatabadi, Sarah Erfani and Christopher Leckie. 'ℓ_∞-Robustness and Beyond: Unleashing Efficient Adversarial Training'. In: *18th European Conference on Computer Vision*. 2022.

[21] Ian Goodfellow, Jonathon Shlens and Christian Szegedy. 'Explaining and Harnessing Adversarial Examples'. In: *International Conference on Learning Representations*. 2015.

[22] Ian Goodfellow et al. 'Generative Adversarial Nets'. In: *Advances in Neural Information Processing Systems*. 2014.

[23] Sven Gowal et al. 'Improving Robustness using Generated Data'. In: *Advances in Neural Information Processing Systems*. 2021.

[24] Sven Gowal et al. *Uncovering the Limits of Adversarial Training against Norm-Bounded Adversarial Examples*. 2020. arXiv: `2010.03593`.

[25] Shixiang Gu and Luca Rigazio. 'Towards Deep Neural Network Architectures Robust to Adversarial Examples'. In: *Workshop Track of the International Conference on Learning Representations*. 2015.

[26] Irina Higgins et al. 'beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework'. In: *International Conference on Learning Representations*. 2017.

[27] Mitch Hill, Jonathan Mitchell and Song-Chun Zhu. 'Stochastic Security: Adversarial Defense Using Long-Run Dynamics of Energy-Based Models'. In: *International Conference on Learning Representations*. 2021.

[28] Chin-Wei Huang, Jae Hyun Lim and Aaron C Courville. 'A Variational Perspective on Diffusion-Based Generative Models and Score Matching'. In: *Advances in Neural Information Processing Systems*. 2021.

[29] Uiwon Hwang et al. 'PuVAE: A Variational Autoencoder to Purify Adversarial Examples'. In: *IEEE Access*. 2019.

[30] Andrew Ilyas et al. 'Adversarial Examples Are Not Bugs, They Are Features'. In: *Advances in Neural Information Processing Systems*. 2019.

[31] Xiaojun Jia et al. 'LAS-AT: Adversarial Training With Learnable Attack Strategy'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022.

[32] Diederik P. Kingma and Max Welling. 'Auto-Encoding Variational Bayes'. In: *International Conference on Learning Representations*. 2014.

[33] Alex Krizhevsky. 'Learning Multiple Layers of Features from Tiny Images'. In: 2009.

[34] Alexey Kurakin, Ian J. Goodfellow and Samy Bengio. 'Adversarial Examples in the Physical World'. In: *Artificial Intelligence Safety and Security* (2018).

[35] Cassidy Laidlaw, Sahil Singla and Soheil Feizi. 'Perceptual Adversarial Robustness: Defense Against Unseen Threat Models'. In: *International Conference on Learning Representations*. 2021.

[36] Yann LeCun and Corinna Cortes. *The MNIST handwritten digit database*. 2010.

[37] Yann LeCun et al. 'A tutorial on energy-based learning'. In: *Predicting structured data*. MIT Press, 2006. Chap. 1.

[38] Minjong Lee and Dongwoo Kim. 'Robust Evaluation of Diffusion-Based Adversarial Purification'. In: *International Conference on Computer Vision*. 2024.

[39] Fangzhou Liao et al. 'Defense Against Adversarial Attacks Using High-Level Representation Guided Denoiser'. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[40] Guang Lin et al. *Robust Diffusion Models for Adversarial Purification*. 2024. arXiv: 2403.16067.

[41] Liyuan Liu et al. 'On the Variance of the Adaptive Learning Rate and Beyond'. In: *International Conference on Learning Representations*. 2020.

[42] Aleksander Madry et al. 'Towards Deep Learning Models Resistant to Adversarial Attacks'. In: *International Conference on Learning Representations*. 2018.

[43] Tom M. Mitchell. *The Need for Biases in Learning Generalizations*. Tech. rep. New Brunswick, NJ: Rutgers University, 1980.

[44] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi and Pascal Frossard. 'DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks'. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[45] Weili Nie et al. 'Diffusion Models for Adversarial Purification'. In: *Proceedings of the International Conference on Machine Learning*. 2022.

[46] Adam Paszke et al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems*. 2019.

[47] ShengYun Peng et al. *Robust Principles: Architectural Design Principles for Adversarially Robust CNNs*. 2023.

[48] Yao Qin et al. 'Imperceptible, Robust, and Targeted Adversarial Examples for Automatic Speech Recognition'. In: *Proceedings of the International Conference on Machine Learning*. 2019.

[49] Sylvestre-Alvise Rebuffi et al. 'Data Augmentation Can Improve Robustness'. In: *Advances in Neural Information Processing Systems*. 2021.

[50] Danilo Jimenez Rezende, Shakir Mohamed and Daan Wierstra. 'Stochastic Backpropagation and Approximate Inference in Deep Generative Models'. In: *Proceedings of the International Conference on Machine Learning*. 2014.

[51] Herbert Robbins and Sutton Monro. 'A Stochastic Approximation Method'. In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407.

[52] Pouya Samangouei, Maya Kabkab and Rama Chellappa. 'Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models'. In: *International Conference on Learning Representations*. 2018.

[53] Changhao Shi, Chester Holtz and Gal Mishne. 'Online Adversarial Purification based on Self-supervised Learning'. In: *International Conference on Learning Representations*. 2021.

[54] Naman D Singh, Francesco Croce and Matthias Hein. *Revisiting Adversarial Training for ImageNet: Architectures, Training and Generalization across Threat Models*. 2023. arXiv: 2303.01870.

[55] Leslie N. Smith. 'Cyclical Learning Rates for Training Neural Networks'. In: *IEEE Winter Conference on Applications of Computer Vision*. 2017.

[56] Kihyuk Sohn, Honglak Lee and Xinchen Yan. 'Learning Structured Output Representation using Deep Conditional Generative Models'. In: *Advances in Neural Information Processing Systems*. 2015.

[57] Christian Szegedy et al. 'Intriguing properties of neural networks'. In: *International Conference on Learning Representations*. 2014.

[58] Florian Tramèr and Dan Boneh. 'Adversarial Training and Robustness for Multiple Perturbations'. In: *Advances in Neural Information Processing Systems*. 2019.

[59] Florian Tramèr et al. 'Ensemble Adversarial Training: Attacks and Defenses'. In: *International Conference on Learning Representations*. 2018.

[60] Florian Tramèr et al. 'On Adaptive Attacks to Adversarial Example Defenses'. In: *Advances in Neural Information Processing Systems*. 2020.

[61] Pascal Vincent et al. 'Extracting and composing robust features with denoising autoencoders'. In: *International Conference on Machine Learning*. 2008.

[62] Zekai Wang et al. *Better Diffusion Models Further Improve Adversarial Training*. 2023. arXiv: 2303.10130.

[63] Han Xiao, Kashif Rasul and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. arXiv: 1708.07747.

[64] Xinchen Yan et al. 'Attribute2Image: Conditional Image Generation from Visual Attributes'. In: *Proceedings of the European Conference on Computer Vision*. 2016.

[65] Zhaoyuan Yang et al. 'Adversarial Purification with the Manifold Hypothesis'. In: *AAAI Conference on Artificial Intelligence*. 2024.

[66] Jongmin Yoon, Sung Ju Hwang and Juho Lee. 'Adversarial Purification with Score-based Generative Models'. In: *Proceedings of the International Conference on Machine Learning*. 2021.

[67] Hongyang Zhang et al. 'Theoretically Principled Trade-off between Robustness and Accuracy'. In: *Proceedings of the International Conference on Machine Learning*. 2019.

[68] M. Zhang, S. Levine and C. Finn. 'MEMO: Test Time Robustness via Adaptation and Augmentation'. In: *Advances in Neural Information Processing Systems*. 2022.

[69] Michael Zhang et al. 'Lookahead Optimizer: k steps forward, 1 step back'. In: *Advances in Neural Information Processing Systems*. 2019.

[70] Zhaoyu Zhang, Mengyan Li and Jun Yu. 'On the Convergence and Mode Collapse of GAN'. In: *SIGGRAPH Asia 2018 Technical Briefs*. 2018.

A On Projected Gradient Descent adversarial training

The task of determining model parameters $\boldsymbol{\theta}^\star$ that are robust to adversarial perturbations is cast in [42] as a *min-max* optimisation problem seeking to minimise *adversarial risk*, *i.e.*:

$$\boldsymbol{\theta}^\star \approx \hat{\boldsymbol{\theta}}^\star := \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}} \left[\max_{\boldsymbol{\delta}\in\mathbb{S}} \mathcal{L}\left(f\left(\boldsymbol{x}+\boldsymbol{\delta};\boldsymbol{\theta}\right), y\right) \right]$$

where $\mathcal{D}$ is the distribution on the examples $\boldsymbol{x}$ and the corresponding labels y, $f(\cdot;\boldsymbol{\theta})$ is a model with learnable parameters $\boldsymbol{\theta}$, $\mathcal{L}$ is a suitable loss function, and $\mathbb{S}$ is the set of allowed constrained perturbations. In the case of ℓ_p norm-bound perturbations of maximum magnitude ϵ, we can further specify $\mathbb{S} := \{\boldsymbol{\delta} \,|\, \|\boldsymbol{\delta}\|_p \leq \epsilon\}$.

The inner optimisation problem is solved, in [42], by *Projected Gradient Descent* (PGD), an iterative algorithm whose goal is the synthesis of an adversarial perturbation $\hat{\boldsymbol{\delta}} = \boldsymbol{\delta}^{(K)}$ after K *gradient ascent and projection* steps defined as:

$$\boldsymbol{\delta}^{(k+1)} \leftarrow \mathfrak{P}_{\mathbb{S}}\Big(\boldsymbol{\delta}^{(k)} + \alpha \operatorname{sign}\Big(\nabla_{\boldsymbol{\delta}^{(k)}} \mathcal{L}_{ce}(f(\boldsymbol{x}+\boldsymbol{\delta}^{(k)};\boldsymbol{\theta}), y)\Big)\Big)$$

where $\boldsymbol{\delta}^{(0)}$ is randomly sampled within $\mathbb{S}$, α is a hyperparameter (*step size*), $\mathcal{L}_{ce}$ is the cross-entropy function, and $\mathfrak{P}_{\mathbb{A}}$ is the Euclidean projection operator onto set $\mathbb{A}$, *i.e.*:

$$\mathfrak{P}_{\mathbb{A}}(\boldsymbol{a}) := \arg\min_{\boldsymbol{a}'\in\mathbb{A}} ||\boldsymbol{a}-\boldsymbol{a}'||_2 \ .$$

The outer optimisation is carried out by simply training $f(\cdot;\boldsymbol{\theta})$ on the examples found by PGD against the current model parameters – and their original pre-perturbation labels. The overall procedure just described constitutes PGD *adversarial training*.

B On the functioning of (conditional) Variational Autoencoders

Variational autoencoders (*VAE*s) [32, 50] learn from data a generative distribution of the form $p(\boldsymbol{x},\boldsymbol{z}) = p(\boldsymbol{x} \,|\, \boldsymbol{z})p(\boldsymbol{z})$, where the probability density $p(\boldsymbol{z})$ represents a prior over latent variable $\boldsymbol{z}$, and $p(\boldsymbol{x} \,|\, \boldsymbol{z})$ is the likelihood function, which can be used to sample data of interest $\boldsymbol{x}$, given $\boldsymbol{z}$.

Training is carried out by maximising a variational lower bound, $-\mathcal{L}_{\text{VAE}}(\boldsymbol{x})$, on the log-likelihood $\log p(\boldsymbol{x})$ – which is a proxy for the *Evidence Lower Bound* (*ELBO*) – *i.e.*:

$$-\mathcal{L}_{\text{VAE}}(\boldsymbol{x}) := \mathbb{E}_{q(\boldsymbol{z}\,|\,\boldsymbol{x})}[\log p(\boldsymbol{x} \,|\, \boldsymbol{z})] - \text{KL}(q(\boldsymbol{z} \,|\, \boldsymbol{x}) \| p(\boldsymbol{z}))$$

where $q(\boldsymbol{z} \,|\, \boldsymbol{x}) \approx p(\boldsymbol{z} \,|\, \boldsymbol{x})$ is an approximate posterior and $\text{KL}(\cdot\|\cdot)$ is the Kullback-Leibler divergence.

By parameterising the likelihood with a *decoder ANN* $p_{\boldsymbol{\theta}_\text{D}}(\boldsymbol{x} \,|\, \boldsymbol{z};\boldsymbol{\theta}_\text{D}) \approx p(\boldsymbol{x} \,|\, \boldsymbol{z})$, and a possible variational posterior with an *encoder ANN* $q_{\boldsymbol{\theta}_\text{E}}(\boldsymbol{z} \,|\, \boldsymbol{x};\boldsymbol{\theta}_\text{E}) \approx q(\boldsymbol{z} \,|\, \boldsymbol{x})$, the parameters $\boldsymbol{\theta}^\star_\text{D}$ of the generative model that best reproduces the data can be learnt – jointly with $\boldsymbol{\theta}^\star_\text{E}$ – as:

$$\begin{aligned}
&\boldsymbol{\theta}^\star_\text{E}, \boldsymbol{\theta}^\star_\text{D} := \\
&\quad \arg\min_{(\boldsymbol{\theta}_\text{E},\boldsymbol{\theta}_\text{D})} \mathcal{L}_{\text{VAE}}(\boldsymbol{x}) = \\
&\quad \arg\min_{(\boldsymbol{\theta}_\text{E},\boldsymbol{\theta}_\text{D})} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}} \left[-\mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\theta}_\text{E}}(\boldsymbol{z}\,|\,\boldsymbol{x};\boldsymbol{\theta}_\text{E})} \left[\log p_{\boldsymbol{\theta}_\text{D}}(\boldsymbol{x} \,|\, \boldsymbol{z};\boldsymbol{\theta}_\text{D})\right] + \text{KL}(q_{\boldsymbol{\theta}_\text{E}}(\boldsymbol{z} \,|\, \boldsymbol{x};\boldsymbol{\theta}_\text{E}) \| p(\boldsymbol{z})) \right]
\end{aligned}$$

where $\mathcal{D}$ is the distribution over the (training) examples $\boldsymbol{x}$.

From a practical point of view, optimisation is based on the empirical evaluation of $\mathcal{L}_{\text{VAE}}(\boldsymbol{x};\boldsymbol{\theta})$ on mini-batches of data, with the term $-\mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\theta}_\text{E}}(\boldsymbol{z}\,|\,\boldsymbol{x};\boldsymbol{\theta}_\text{E})} \left[\log p_{\boldsymbol{\theta}_\text{D}}(\boldsymbol{x} \,|\, \boldsymbol{z};\boldsymbol{\theta}_\text{D})\right]$ replaced by a *reconstruction cost*

$$\mathcal{L}_{\text{Reco}}(\boldsymbol{x}, \boldsymbol{x'}) \geq 0 \,|\, \mathcal{L}_{\text{Reco}}(\boldsymbol{x}, \boldsymbol{x}') = 0 \iff x = x' \,.$$

The generation of new data according to the fitted model is achieved by sampling from

$$p_{\boldsymbol{\theta}^\star_{\text{D}}}(\boldsymbol{x} \,|\, \boldsymbol{z}; \boldsymbol{\theta}^\star_{\text{D}})\Big|_{\boldsymbol{z} \sim p(\boldsymbol{z})}$$

i.e. decoding samples from $p(\boldsymbol{z})$.

The setting is analogous in the case of *conditional* Variational Autoencoders [56, 64] (see section 3), where conditional sampling is achieved by

$$\boldsymbol{x}_{\boldsymbol{c_j}} \sim p_{\boldsymbol{\theta}^\star_{\text{D}}}(\boldsymbol{x} \,|\, \boldsymbol{z}, \boldsymbol{c}; \boldsymbol{\theta}^\star_{\text{D}})\Big|_{\boldsymbol{z} \sim p(\boldsymbol{z});\ \boldsymbol{c} = \boldsymbol{c_j}} \,.$$

C Justification of Adversarially-balanced batches

During the incipient phases of experimentation, preliminary tests were performed with the MNIST [36] and Fashion-MNIST [63] datasets – using a conditional *VAE* as the *purifier*, and small FCNs or *convolutional ANN*s as the *classifiers*. Adversarial examples were generated against the adversarially pre-trained *classifier*, and tentatively denoised by the *purifier* with one sample only. The resulting recovered inputs were classified by the *classifier* and the overall accuracy was recorded.

Importantly, such tests were not meant to assess the *end-to-end* adversarial robustness of the whole architecture, but only to tune the training protocol of the *purifier*.

Generating adversarial training examples by means of PGD is considered the *gold standard* [24] and was first attempted as a natural choice to train the purifier. However, in this case, the following phenomena were observed:

- Unsatisfactory *clean* accuracy was reached upon convergence, speculatively a consequence of the *VAE* having never been trained on *clean*-to-*clean* example reconstruction;
- Persistent vulnerability to same norm-bound FGSM perturbations was noticed;
- Persistent vulnerability to smaller norm-bound FGSM and PGD perturbations was noticed.

In an attempt to mitigate such issues, the composition of adversarial examples was adjusted to specifically counteract each of the issues uncovered. The adoption of any smaller subset of attack types or strength, compared to that described in subsection 4.4, resulted in unsatisfactory mitigation.

At that point, another problem emerged: if such an adversarial training protocol was carried out in homogeneous batches, each containing the same type and strength of attack (or none at all), the resulting robust accuracy was still partially compromised due to the homogeneous ordering of attack types and strengths across batches.

Such observations lead to the final formulation of the training protocol, detailed in subsection 4.4, which mitigates to the best the issues described so far.

D Heuristic justification of the *robust aggregation strategy*

The rationale leading to the choice of the specific *robust aggregation strategy* described in subsection 4.5 was an attempt to answer the following question: 'How is it possible to aggregate the results of an ensemble of classifiers in a way such that it is hard to tilt the balance of the ensemble by attacking only a few of its members?'. The same reasoning can be extended to the *reciprocal* problem we are trying to solve here, where different input reconstructions obtained from the same potentially perturbed input are classified by the same model (the *classifier*).

Far from providing a satisfactory answer, we can analyse the behaviour of our aggregation strategy as the logit associated with a given model and class varies across its domain, under the effect of

adversarial intervention. Comparison with existing (and more popular) *probability averaging* and *logit averaging* aggregation strategies should provide a heuristic justification of our choice.

We recall our aggregation strategy:

$$P_i := \frac{1}{Z} \prod_{\alpha=1}^{N} e^{e^{l_i^\alpha}}.$$

Additionally, we recall *logit averaging* aggregation

$$P_i := \frac{1}{Z} e^{\frac{1}{N}\sum_{\alpha=1}^{N} l_i^\alpha} = \frac{1}{Z} \prod_{\alpha=1}^{N} e^{\frac{1}{N} l_i^\alpha} = \frac{1}{Z} \left(\prod_{\alpha=1}^{N} e^{l_i^\alpha} \right)^{\frac{1}{N}}$$

and *probability averaging* aggregation

$$P_i := \frac{1}{Z} \sum_{\alpha=1}^{N} \frac{e^{l_i^\alpha}}{\sum_{j=1}^{C} e^{l_j^\alpha}} = \sum_{\alpha=1}^{N} e^{l_i^\alpha} \frac{1}{Q^\alpha}$$

where $Q^\alpha = \sum_{j=1}^{C} e^{l_j^\alpha}$.

Finally, since $l_i^\alpha \in \mathbb{R}, \forall l_i^\alpha$, $\lim_{x \to -\infty} e^x = 0$ and $e^0 = 1$, we can observe that $e^{l_i^\alpha} > 0$ and $e^{e^{l_i^\alpha}} > 1, \forall l_i^\alpha$.

Now, we consider a given class $i^\star$ and the *classifier* prediction on a given input reconstruction $\alpha^\star$, and study the potential effect of an adversary acting on $l_{i^\star}^{\alpha^\star}$. This adversarial intervention can be framed in two complementary scenarios: either the class $i^\star$ is correct and the adversary aims to decrease its membership probability, or the class $i^\star$ is incorrect and the adversary aims to increase its membership probability. In any case, the adversary should comply with the ϵ_∞-boundedness of its perturbation on the input.

Logit averaging In the former scenario, the product of $e^{l_i^\alpha}$ terms can be arbitrarily deflated (up to zero) by lowering the $l_{i^\star}^{\alpha^\star}$ logit only. In the latter scenario, the logit can be arbitrarily inflated, and such effect is only partially suppressed by normalisation by Z (a sum of $1/N$-exponentiated terms).

Probability averaging In the former scenario, although the effect of the deflation of a single logit is bounded by $e^{l_{i^\star}^{\alpha^\star}} > 0$, two attack strategies are possible: either decreasing the value of $l_{i^\star}^{\alpha^\star}$ or increasing the value of $Q^{\alpha^\star}$, giving rise to complex combined effects. In the latter scenario, the reciprocal is possible, *i.e.* either inflating $l_{i^\star}^{\alpha^\star}$ or deflating $Q^{\alpha^\star}$. Normalisation has no effect in both cases.

Ours In the former scenario, the effect of logit deflation on a single product term is bounded by $e^{e^{l_{i^\star}^{\alpha^\star}}} > 1$, thus exerting only a minimal collateral effect on the product, through a decrease of Z. This effectively prevents *aggregation takeover* by logit deflation. Similarly to *logit averaging*, in the latter scenario, the logit can be arbitrarily inflated. However, in this case, the effect of normalisation by Z is much stronger, given its increased magnitude.

From such a comparison, our aggregation strategy is the only one that strongly prevents *adversarial takeover* by *logit deflation*, while still defending well against perturbations targeting *logit inflation*.

E Architectural details and hyperparameters

In the following section, we provide more precise details about the architectures (subsection E.1) and hyperparameters (subsection E.2) used in the experimental phase of our work.

E.1 Architectures

In the following subsection, we describe the specific structure of the individual parts composing the *purifier* – in the three scenarios considered. As far as the *classifier* architectures are concerned, we redirect the reader to the original articles introducing those models (*i.e.*: [18] for *scenarios (a)* and *(b)*, [62] for *scenario (c)*).

During training, before being processed by the *purifier* encoder, input examples are standardised according to the statistics of the respective training dataset.

Afterwards, they are fed to the disjoint input encoder (see subsection 4.3), whose architecture is shown in Table 4. The same architecture is used in all scenarios considered.

Table 4: Architecture for the *disjoint input encoder* of the *purifier*. The same architecture is used in all scenarios considered. The architecture is represented layer by layer, from input to output, in a PyTorch-like syntax. The following abbreviations are used: `Conv2D`: 2-dimensional convolutional layer; `ch_in`: number of input channels; `ch_out`: number of output channels; `ks`: kernel size; `s`: stride; `p`: padding; `b`: presence of a learnable bias term; `BatchNorm2D`: 2-dimensional batch normalisation layer; `affine`: presence of learnable affine transform coefficients; `slope`: slope for the activation function in the negative semi-domain.

Disjoint Input Encoder (all scenarios)
`Conv2D(ch_in=3, ch_out=6, ks=3, s=2, p=1, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`Conv2D(ch_in=6, ch_out=12, ks=3, s=2, p=1, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`

The original input is also fed to the *classifier*. The corresponding internal representation is extracted, preserving its layered structure. In order to improve the scalability of the method, only a subset of *classifier* layers is used instead of the whole internal representation. Specifically, for each *block* of the WIDERESNET architecture, only the first layers have been considered; two *shortcut layers* have also been added for good measure. The exact list of those layers is reported in Table 5.

Each extracted layerwise (pre)activation tensor has the shape of a multi-channel image, which is processed – independently for each layer – by a different CNN whose individual architecture is shown in Table 6 (*scenarios (a)* and *(b)*) and Table 7 (*scenario (c)*).

The resulting tensors (still having the shape of multi-channel images) are then jointly processed by a fully-connected subnetwork whose architecture is shown in Table 8. The value of `fcrepr` for the different scenarios considered is shown in Table 13.

The *compressed input* and *compressed internal representation* so obtained are finally jointly encoded by an additional fully-connected subnetwork whose architecture is shown in Table 9. The output is a tuple of means and standard deviations to be used to sample the stochastic latent code $\boldsymbol{z}$.

The sampler used for the generation of such latent variables $\boldsymbol{z}$, during the training of the *purifier*, is a reparameterised [32] Normal sampler $\boldsymbol{z} \sim \mathcal{N}(\mu, \sigma)$. During inference, $\boldsymbol{z}$ is sampled by reparameterisation from the *i.i.d* Standard Normal distribution $\boldsymbol{z} \sim \mathcal{N}(0, 1)$ (*i.e.* from its original prior).

The architectures for the decoder of the *purifier* are shown in Table 10 (*scenarios (a)* and *(b)*) and Table 11 (*scenario (c)*).

E.2 Hyperparameters

In the following section, we provide the hyperparameters used for *adversarial example generation* and *optimisation* during the training of the *purifier*, and those related to the *purifier* model architectures. We also provide the hyperparameters for the PGD+EOT attack, which is used as a complementary tool for the evaluation of adversarial robustness.

Table 5: *Classifier* model (WIDERESNET-28-10) layer names used as (a subset of) the *internal representation* fed to the *layerwise convolutional encoder* of the *purifier*. The names reflect those used in the model implementation.

All scenarios
`layer.0.block.0.conv_0`
`layer.0.block.0.conv_1`
`layer.0.block.1.conv_0`
`layer.0.block.1.conv_1`
`layer.0.block.2.conv_0`
`layer.0.block.2.conv_1`
`layer.0.block.3.conv_0`
`layer.0.block.3.conv_1`
`layer.1.block.0.conv_0`
`layer.1.block.0.conv_1`
`layer.1.block.0.shortcut`
`layer.1.block.1.conv_0`
`layer.1.block.1.conv_1`
`layer.1.block.2.conv_0`
`layer.1.block.2.conv_1`
`layer.1.block.3.conv_0`
`layer.1.block.3.conv_1`
`layer.2.block.0.conv_0`
`layer.2.block.0.conv_1`
`layer.2.block.0.shortcut`
`layer.2.block.1.conv_0`
`layer.2.block.1.conv_1`
`layer.2.block.2.conv_0`
`layer.2.block.2.conv_1`
`layer.2.block.3.conv_0`
`layer.2.block.3.conv_1`

Table 6: Architecture for the *layerwise internal representation encoder* of the *purifier*. The architecture shown in this table is used in *scenarios (a)* and *(b)*. The architecture is represented layer by layer, from input to output, in a PyTorch-like syntax. The following abbreviations are used: `Conv2D`: 2-dimensional convolutional layer; `ch_in`: number of input channels; `ch_out`: number of output channels; `ks`: kernel size; `s`: stride; `p`: padding; `b`: presence of a learnable bias term; `BatchNorm2D`: 2-dimensional batch normalisation layer; `affine`: presence of learnable affine transform coefficients; `slope`: slope for the activation function in the negative semi-domain. The abbreviation `[ci]` indicates the number of input channels for the *(pre)activation tensor* of each extracted layer. The abbreviation `ceil` indicates the *ceiling* integer rounding function.

Layerwise Internal Representation Encoder (scenarios (a) and (b))
`Conv2D(ch_in=[ci], ch_out=ceil([ci]/2), ks=3, s=1, p=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`Conv2D(ch_in=ceil([ci]/2), ch_out=ceil([ci]/4), ks=3, s=1, p=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`Conv2D(ch_in=ceil([ci]/4), ch_out=ceil([ci]/8), ks=3, s=1, p=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`

Table 7: Architecture for the *layerwise internal representation encoder* of the *purifier*. The architecture shown in this table is used in *scenario (c)*. The architecture is represented layer by layer, from input to output, in a PyTorch-like syntax. The following abbreviations are used: `Conv2D`: 2-dimensional convolutional layer; `ch_in`: number of input channels; `ch_out`: number of output channels; `ks`: kernel size; `s`: stride; `p`: padding; `b`: presence of a learnable bias term; `BatchNorm2D`: 2-dimensional batch normalisation layer; `affine`: presence of learnable affine transform coefficients; `slope`: slope for the activation function in the negative semi-domain. The abbreviation `[ci]` indicates the number of input channels for the *(pre)activation tensor* of each extracted layer. The abbreviation `ceil` indicates the *ceiling* integer rounding function.

Layerwise Internal Representation Encoder (scenario (c))
`Conv2D(ch_in=[ci], ch_out=ceil([ci]/2), ks=3, s=1, p=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`Conv2D(ch_in=ceil([ci]/2), ch_out=ceil([ci]/4), ks=3, s=1, p=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`Conv2D(ch_in=ceil([ci]/4), ch_out=ceil([ci]/8), ks=3, s=1, p=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`Conv2D(ch_in=ceil([ci]/8), ch_out=ceil([ci]/16), ks=3, s=1, p=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`

Table 8: Architecture for the *fully-connected representation encoder* of the *purifier*. The architecture shown in this table is used in all scenarios considered. The architecture is represented layer by layer, from input to output, in a PyTorch-like syntax. The following abbreviations are used: `Concatenate`: layer concatenating its input features; `flatten_features`: whether the input features are to be flattened before concatenation; `feats_in`, `feats_out`: number of input and output features of a linear layer; `b`: presence of a learnable bias term; `BatchNorm1D`: 1-dimensional batch normalisation layer; `affine`: presence of learnable affine transform coefficients; `slope`: slope for the activation function in the negative semi-domain. The abbreviation `[computed]` indicates that the number of features is computed according to the shape of the concatenated input tensors. The value of `fcrepr` for the different scenarios considered is shown in Table 13.

Fully-Connected Representation Encoder (all scenarios)
`Concatenate(flatten_features=True)`
`Linear(feats_in=[computed], feats_out=fcrepr, b=False)`
`BatchNorm1D(affine=True)`
`LeakyReLU(slope=0.2)`

Table 9: Architecture for the *fully-connected joint encoder* of the *purifier*. The architecture shown in this table is used in all scenarios considered. The architecture is represented layer by layer, from input to output, in a PyTorch-like syntax. The following abbreviations are used: `Concatenate`: layer concatenating its input features; `flatten_features`: whether the input features are to be flattened before concatenation; `feats_in`, `feats_out`: number of input and output features of a linear layer; `b`: presence of a learnable bias term; `BatchNorm1D`: 1-dimensional batch normalisation layer; `affine`: presence of learnable affine transform coefficients; `slope`: slope for the activation function in the negative semi-domain. The abbreviation `[computed]` indicates that the number of features is computed according to the shape of the concatenated input tensors. The value of `fjoint` for the different scenarios considered is shown in Table 13. The last *layer* of the network returns a tuple of 2 tensors, each independently processed – from the output of the previous layer – by the two comma-separated *sub-layers*.

Fully-Connected Joint Encoder (all scenarios)
`Concatenate(flatten_features=True)`
`Linear(feats_in=[computed], feats_out=fjoint, b=False)`
`BatchNorm1D(affine=True)`
`LeakyReLU(slope=0.2)`
`( Linear(feats_in=fjoint, feats_out=fjoint, b=True),`
`  Linear(feats_in=fjoint, feats_out=fjoint, b=True) )`

Table 10: Architecture for the decoder of the *purifier*. The architecture shown in this table is used in *scenarios (a)* and *(b)*. The architecture is represented layer by layer, from input to output, in a PyTorch-like syntax. The following abbreviations are used: `Concatenate`: layer concatenating its input features; `flatten_features`: whether the input features are to be flattened before concatenation; `feats_in`, `feats_out`: number of input and output features of a linear layer; `b`: presence of a learnable bias term; `ConvTranspose2D`: 2-dimensional transposed convolutional layer; `ch_in`: number of input channels; `ch_out`: number of output channels; `ks`: kernel size; `s`: stride; `p`: padding; `op`: PyTorch parameter '`output padding`', used to disambiguate the number of spatial dimensions of the resulting output; `b`: presence of a learnable bias term; `BatchNorm2D`: 2-dimensional batch normalisation layer; `affine`: presence of learnable affine transform coefficients; `slope`: slope for the activation function in the negative semi-domain. The values of `fjoint` and `fcrepr` for the different scenarios considered are shown in Table 13.

Decoder (scenarios (a) and (b))
`Concatenate(flatten_features=True)`
`Linear(feats_in=[fjoint+fcrepr], feats_out=2304, b=True)`
`LeakyReLU(slope=0.2)`
`Unflatten(256, 3, 3)`
`ConvTranspose2D(ch_in=256, ch_out=256, ks=3, s=2, p=1, op=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`ConvTranspose2D(ch_in=256, ch_out=128, ks=3, s=2, p=1, op=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`ConvTranspose2D(ch_in=128, ch_out=64, ks=3, s=2, p=1, op=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`ConvTranspose2D(ch_in=64, ch_out=32, ks=3, s=2, p=1, op=0, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`ConvTranspose2D(ch_in=32, ch_out=3, ks=2, s=1, p=1, op=0, b=True)`
`Sigmoid()`

Table 11: Architecture for the decoder of the *purifier*. The architecture shown in this table is used in *scenario (c)*. The architecture is represented layer by layer, from input to output, in a PyTorch-like syntax. The following abbreviations are used: `Concatenate`: layer concatenating its input features; `flatten_features`: whether the input features are to be flattened before concatenation; `feats_in`, `feats_out`: number of input and output features of a linear layer; `b`: presence of a learnable bias term; `ConvTranspose2D`: 2-dimensional transposed convolutional layer; `ch_in`: number of input channels; `ch_out`: number of output channels; `ks`: kernel size; `s`: stride; `p`: padding; `op`: PyTorch parameter '`output padding`', used to disambiguate the number of spatial dimensions of the resulting output; `b`: presence of a learnable bias term; `BatchNorm2D`: 2-dimensional batch normalisation layer; `affine`: presence of learnable affine transform coefficients; `slope`: slope for the activation function in the negative semi-domain. The values of `fjoint` and `fcrepr` for the different scenarios considered are shown in Table 13.

Decoder (scenario (c))
`Concatenate(flatten_features=True)`
`Linear(feats_in=[fjoint+fcrepr], feats_out=4096, b=True)`
`LeakyReLU(slope=0.2)`
`Unflatten(256, 4, 4)`
`ConvTranspose2D(ch_in=256, ch_out=256, ks=3, s=2, p=1, op=1, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`ConvTranspose2D(ch_in=256, ch_out=128, ks=3, s=2, p=1, op=1, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`ConvTranspose2D(ch_in=128, ch_out=64, ks=3, s=2, p=1, op=1, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`ConvTranspose2D(ch_in=64, ch_out=32, ks=3, s=2, p=1, op=1, b=False)`
`BatchNorm2D(affine=True)`
`LeakyReLU(slope=0.2)`
`ConvTranspose2D(ch_in=32, ch_out=3, ks=3, s=1, p=1, op=0, b=True)`
`Sigmoid()`

Attacks The hyperparameters used for the adversarial attacks described in subsection 4.4 are shown in Table 12. The value of ϵ_∞ is fixed to $\epsilon_\infty = 8/255$. With the only exception of ϵ_∞, AUTOATTACK is to be considered a *hyperparameter-free* adversarial example generator.

Table 12: Hyperparameters for the attacks used for training and testing the *purifier* The FGSM and PDG attacks refer to the training phase (see subsection 4.4), whereas the PGD+EOT attack [38] refers to the robustness assessment pipeline. The entry `CCE` denotes the *Categorical CrossEntropy* loss function. The ℓ_∞ threat model is assumed, and all inputs are linearly rescaled within $[0.0, 1.0]$ before the attack.

	FGSM	PGD	PGD+EOT
Input clipping	$[0.0, 1.0]$	$[0.0, 1.0]$	$[0.0, 1.0]$
# of steps	1	40	200
Step size	ϵ_∞	0.01	0.007
Loss function	*CCE*	*CCE*	*CCE*
# of EoT iterations	1	1	20
Optimiser		*SGD*	*SGD*

Architectures Table 13 contains the hyperparameters that define the model architectures used as part of the *purifier*, in the different scenarios considered.

Table 13: Scenario-specific architectural hyperparameters for the *purifier*, as referred to in Table 8, Table 9, Table 10, and Table 11.

	Scenario (a)	*Scenario (b)*	*Scenario (c)*
`fcrepr`	512	512	768
`fjoint`	128	128	192

Training Table 14 collects the hyperparameters governing the training of the *purifier* in the different scenarios considered.

Table 14: Hyperparameters used for training the *purifier*, grouped by scenario. The entry `CCE` denotes the *Categorical CrossEntropy* loss function. The *LR* scheduler is stepped after each epoch.

	All *scenarios*	*Sc. (a)*	*Sc. (b)*	*Sc. (c)*
Optimiser	RADAM+LOOKAHEAD			
RADAM β_1	0.9			
RADAM β_2	0.999			
RADAM ϵ	10^{-8}			
RADAM *Weight Decay*	None			
LOOKAHEAD *averaging decay*	0.8			
LOOKAHEAD steps	6			
Initial *LR*	5×10^{-9}			
Loss function	*CCE*			
Sampled reconstructions per input	8			
Epochs		200	200	250
LR warm-up epochs		25	25	31
LR plateau epochs		25	25	31
LR annealing epochs		150	250	188
Plateau *LR*		0.064	0.064	0.0128
Final *LR*		4.346×10^{-4}	4.346×10^{-4}	1.378×10^{-4}
β *increase* initial epoch		25	25	32
β *increase* final epoch		34	34	43
Batch size		5120	2560	1024
Adversarial *batch fraction*		0.5	0.15	0.01

F Additional tables

The following section contains additional tabular data that may be of interest to the reader.

Table 15 reports the respective clean accuracies for the best models available in terms of AUTOATTACK robust accuracy, in *scenarios (a)* and *(b)*. Models are further divided in AT-based and purification-based, so as to match the corresponding columns for robust accuracy shown in Table 2.

The best AT-based model for CIFAR-10 is taken from [18], whereas that for CIFAR-100 from [62]. Both best purification-based models are taken from [40].

The clean and robust accuracies for the best AT-based model on TINYIMAGENET-200 (*scenario (c)*) are already part of Table 2 and we redirect the reader there for such information. We are not aware of any published *state-of-the-art* adversarial purification-based model for TINYIMAGENET-200.

Table 15: Clean accuracy for the best models (by robust accuracy) on the datasets considered in *scenarios (a)* and *(b)*, mentioned in Table 2. The following abbreviations are used: `Scen`: scenario considered; `Best AT/Cl`: clean accuracy for the most robust model (by the means of AUTOATTACK) on the respective dataset, obtained by adversarial training alone; `Best P/Cl`: clean accuracy for the most robust model (by the means of AUTOATTACK) on the respective dataset, obtained by adversarial purification alone.

Scen.	Dataset	Best AT/Cl	Best P/Cl
(a)	CIFAR-10	0.9323	0.9082
(b)	CIFAR-100	0.7522	0.6973

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Claims made in the abstract and introduction of the paper accurately reflect its contributions, and those are directly corroborated by experimental analysis. Results and their discussion is available in subsection 5.2 and subsection 5.3.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: subsection 5.2 and subsection 5.3 also contain the discussion of potential limitations of the method, and open problems it introduces.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: The paper does not contribute novel theoretical results. Assumptions anyway related to the contribution are clearly stated throughout the paper.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Training and evaluation details required to reproduce the experimental results of the paper are reported in section 5 and Appendix E. Code and data for the reproduction of all experiments are additionally released as part of Supplementary Material.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code and data for the reproduction of all experiments are released to reviewers as part of Supplementary Material. The public version of the paper will include instructions to obtain the same material from a dedicated publicly accessible source.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimiser, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Training and evaluation details, including hyperparameters, required to reproduce the experimental results of the paper are reported in section 5 and Appendix E. Code and data for the reproduction of all experiments are additionally released as part of Supplementary Material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: As doing adversarial training with 40-steps PGD is roughly 40 times more computationally demanding than nominal training, unfortunately, we are unable to show error bars or otherwise quantify statistical errors. In any case, the improvement induced by our method *w.r.t.* their *state-of-the-art* counterparts is well clear of the threshold for statistical significance.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Information about computational resources and required time is contained in subsection 5.1 as well as in the supplementary materials.

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper does conform, in every respect, with the NeurIPS Code of Ethics.

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper proposes a novel technique to mitigate the problem of adversarial vulnerability of high-dimensional classifiers. Such vulnerability may pose potential societal impacts, as discussed in section 1.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We do not plan to release models or data with a high risk for misuse. Models to be released do not reasonably carry risk for misuse.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The creators and/or owners of assets used in the paper are either credited by reference to their original research work, or directly with a link to the preferred landing page for such assets. The use of licensed material is compliant with the respective licenses.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Trained model weights are provided alongside the paper, and their details are provided as part of supplementary materials. In case of release to the public, such details will be provided contextually to the models.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: No crowdsourcing or research with human subjects has been performed as part of the work of, or leading to, this paper.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: No crowdsourcing or research with human subjects has been performed as part of the work of or leading to this paper - including that potentially requiring IRB approval or equivalent authorisation.