NOISY ADVERSARIAL TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

In image classification, data augmentation and the usage of additional data has been shown to increase the efficiency of clean training and the accuracy of the resulting model. However, this does not prevent models from being fooled by adversarial manipulations. To increase the robustness, Adversarial Training (AT) is an easy, yet effective and widely used method to harden neural networks against adversarial inputs. Still, AT is computationally expensive and inefficient in that way, that only one adversarial input per sample of the current batch is created. We propose Noisy Adversarial Training (N-AT), which, for the first time, combines data augmentation in the decision space and adversarial training. By adding random noise to the original adversarial output vector, we create multiple pseudo adversarial instances, thus increasing the data pool for adversarial training. We show that this general idea is applicable to two different learning paradigms, i.e., supervised and self-supervised learning. Using N-AT instead of AT, we can increase the robustness relatively by 1.06% for small seen attacks. For larger seen attacks, the relative gain in robustness increases up to 89.26%. When combining a larger corpus of input data with our proposed method, we report an increase of the clean accuracy and for all observed attacks, compared to AT. In self-supervised training, we observe a similar increase in robust accuracy for seen attacks and large unseen attacks, when it comes to the downstream task of image classification. In addition, when the pretrained model is finetuned, we also report a relative gain in clean accuracy between 0.5% and 1.11%.

1 INTRODUCTION

The performance of deep learning models in various domains, e.g., image classification (Zhai et al. (2021)), semantic image segmentation (Tao et al. (2020)), or reinforcement learning (Tang et al. (2017)) is already on a high level and constantly improving. Among other aspects, ongoing research and advances in data augmentation (Cubuk et al. (2020)) techniques, as well as the creation of more realistic synthetic inputs (Ho et al. (2020)) contribute to this success. Both techniques aim to enrich the training data and increase performance. However, when it comes to safety-critical applications, e.g., autonomous driving, adversarial inputs pose a threat. By applying small but malicious manipulations to the input, the prediction of the model can change drastically.

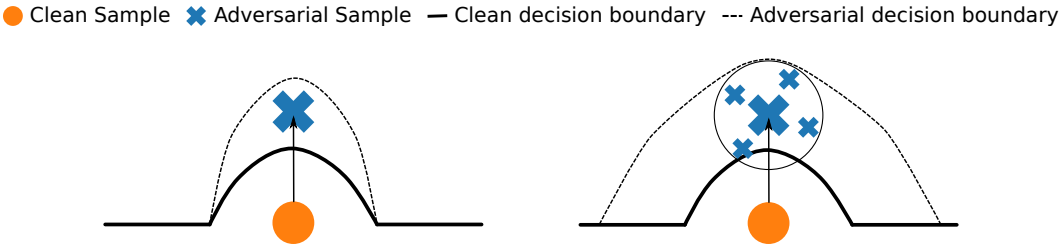
Starting by manipulating digital inputs, several authors, e.g., Goodfellow et al. (2014); Carlini & Wagner (2017); Madry et al. (2017), developed different techniques to calculate and create the necessary manipulations to fool neural networks into misclassifying a given input. Later, these attacks were adapted or extended to also work in the physical world (Athalye et al. (2018); Worzyk et al. (2019); Ranjan et al. (2019)).

One widely used technique to harden neural networks against such attacks is called Adversarial Training (AT), which is simple but yet very effective. The idea is to create adversarial inputs during the training process, and include or exclusively use them for training. Thereby, the model learns to be more resilient against these worst-case perturbations. Madry et al. (2017) for example, proposed a method referred to as Projected Gradient Descent (PGD) which is very successful in finding adversarial inputs and furthermore use these adversarial instances exclusively for training a given model. Even though effective, this form of AT is rather inefficient in that way, that only one adversarial input is created per sample in the current batch.

To create a more efficient way of AT, we propose to **extend adversarial training by data augmentation in the decision space and** to increase the impact, and thereby the efficiency of any given adversarial instance during adversarial training. While data augmentation in the input space is widely used and researched, to the best of our knowledge, data augmentation in the decision space to increase the efficiency of adversarial training has not yet been investigated. The overall concept is shown in Figure 1.

In Figure 1a, the decision space during traditional adversarial training is displayed. Based on the current decision boundary (bold line) and the output vector for a clean sample (orange circle), an adversarial input (blue cross) is created, whose output vector is located on the *wrong* side of the decision boundary. The adversarial decision boundary (dashed line) is then optimized to contribute for the adversarial input.

Figure 1b outlines our extension to this process. Based on the output vector of an adversarial input (large blue cross), multiple pseudo adversarial inputs (small blue crosses) are created by applying normally distributed random noise within a predefined radius to the initial adversarial output vector. By *scattering* the adversarial output vector, we widen its impact on the new adversarial decision boundary (dashed line).



(a) Adversarial Training (b) Noisy Adversarial Training

Figure 1: Difference between traditional (1a) and noisy (1b) adversarial training. Given the output vector of a clean sample (orange circle) and the previous decision boundary (solid line), adversarial inputs (blue cross) lying on the *wrong* side of the decision boundary are created. When adding the adversarial samples to the training process, the decision boundary adapts accordingly (dashed line). During Noisy Adversarial Training, the initial adversarial output vector is perturbed randomly within a given radius to create a set of additional pseudo adversarial inputs (smaller blue crosses). This extends the impact of any single adversarial input on the adversarial decision boundary (dashed line) during training.

The remainder of this paper is structured as follows. In Section 2, we introduce the supervised, as well as self-supervised adversarial training methods used for the experiments in this paper. In Section 3, we define our proposed N-AT method in more detail, followed by the experiments and their discussion in Section 4. Section 5 concludes this paper.

2 Background

2.1 Supervised Adversarial Training

PGD: One of the widest known and applied techniques for adversarial training is PGD, proposed by Madry et al. (2017). Instead of using clean samples during training, the authors use the corresponding adversarial instances, created based on the following iterative equation,

$$x^{i+1} = \Pi_{B(x,\varepsilon)} \left(x^i + \alpha \text{sign} \left(\nabla_{x^i} \mathcal{L}_{\text{CE}} \left(\theta, x^i, y \right) \right) \right) \ , x^0 = x \ . \tag{1}$$

This function is initialized by the original input x and calculates the gradients, regarding the intermediate adversarial input x^i, of the cross-entropy loss $\mathcal{L}_{\text{CE}}$ away from the true label y. This calculation is based on the current parameters of the model θ. The sign of the gradients is multiplied by a step size parameter α and added to the current intermediate adversarial input. The projection Π then limits the perturbation to be within an ε-ball around the initial input x. The parameter ε essentially governs the allowed amount of perturbation. For example, $\varepsilon = 8/255$ regarding the ℓ_∞ norm would

state, that each pixel of the original input vector is allowed to be increased or decreased by not more than 8 values.

2.2 SELF-SUPERVISED ADVERSARIAL TRAINING

More recently, adversarial training is also applied to self-supervised training. The general goal of self-supervision is to train for some pretext tasks where no labels are required. After training, the model and its parameters are transferred to a given downstream task, e.g., image classification. Therefore, the overall model in self-supervised learning is split into a backbone and a projector. The backbone can be based on, e.g., a ResNet architecture (He et al. (2016)), stripped of the last fully connected layer. The projector reduces the dimensionality of the output features from the backbone to a usually 128-dimensional vector. To train without labels, given a batch of samples $\{x_1, ..., x_b\}$, each sample is duplicated and transformed by a given series of random transformations t, e.g., cropping and flipping. The resulting transformed versions of the same origin, i.e., $t_1(x_i)$ and $t_2(x_i)$ are called positive pair, while pairs of samples with different origin, i.e., $t(x_i)$ and $t(x_j)$ with $i \neq j$ are called negative pair. The pretext task of the models used in this paper is to maximise the distance between the output vectors of negative pairs while minimising the distance between the output vectors of positive pairs.

After pretraining for the pretext task, the backbone is kept and the projector is discarded. Instead of the projector, a downstream task-specific head is attached. The parameters of the backbone are usually frozen and only the classification head is trained. In essence, self-supervised pretraining aims to learn good feature representations which can, later on, be used for the given downstream task.

RoCL: Kim et al. (2020) added adversarial training to the SimCLR (Chen et al. (2020)) framework and dubbed it Robust Contrastive Learning (RoCL). To calculate the distance within positive and negative pairs, SimCLR uses the contrastive loss function as given in Table 2 in Appendix A.1 where sim is the cosine similarity. To create the necessary adversarial inputs, Kim et al. (2020) adapted PGD (cf. Equation 1) to use the contrastive loss instead of the cross-entropy loss as follows,

$$t_1(x)^{i+1} = \Pi_{B(t_1(x),\varepsilon)}\left(t_1(x)^i + \alpha \text{sign}\left(\nabla_{t_1(x)^i}\mathcal{L}_{\text{con},\theta}\left(t_1(x)^i, \{t_2(x)\}, \{t_1(x)_{\text{neg}}\}\right)\right)\right), \quad (2)$$

where t_i is a random transformation, while the rest of the notation is the same as in Equation 1. To implement adversarial training, Kim et al. (2020) essentially use the adversarial inputs to extend the positive and negative pairs to triplets. During training, they aim to minimize the distance between the two transformed inputs, as well as the distances between the two transformed inputs each and the created adversarial input for the given pair. The formalization of the RoCL objective is given in Table 2, where $t(x) + \delta$ is the adversarial sample. The overall training loss is then calculated based on the standard contrastive loss only considering the clean transformed samples, plus the adversarial loss based on the triplets of two transformed inputs and the additional adversarial input.

One challenge using SimCLR as the basic framework is that it requires a large batch size to achieve good performance (Chen et al. (2020)). That comes from the fact that no form of dictionary or memory bank is used to increase the number of negative samples, which are essential for good performance in this type of self-supervised learning. By including adversarial samples into the training process, the batch size per GPU has to be reduced compared to standard SimCLR training.

AMOC: Another widely known self-supervised framework is Momentum Contrast (MoCo) proposed by He et al. (2020). The conceptual idea is the same as for SimCLR, i.e., minimising the distance between positive instances while maximising the distance towards negative samples. However, to overcome the problem of large batch sizes, He et al. (2020) implement a dictionary or memory bank and use two networks of the same architecture and the same initial weights. One model is referred to as the query encoder, which is updated after each batch as usual. The other model is called momentum or key encoder, whose parameters are a copy of the query encoder, delayed by a predefined momentum. This makes the output of the key encoder slightly different from the output of the query encoder, which can be considered as an additional form of data augmentation. After processing the inputs of the current batch, the encoded vectors of the momentum encoder are enriched by output vectors calculated during the last batches, stored in the dictionary. Thereby, a large number of negative samples can be created consistently, leading to overall better performance.

The standard loss function for MoCo is given in Table 2, where q and k refer to the encoded vector by the query, resp. key encoder, τ is a temperature parameter, and $\mathcal{M}$ refers to the memory bank of old key vectors.

Based on this framework, Xu & Yang (2020) proposed an extension for adversarial training. They introduce a second memory bank to store exclusively the historic adversarial inputs, and to further disentangle the clean and adversarial distribution, they use dual Batch Normalization as proposed by Xie et al. (2020). The optimization problem they solve is given in Table 2, with t_1 and t_2 being two different random transformations from a set $\mathcal{T}$ of possible transformation, and δ being the adversarial perturbation. $\mathcal{M}_{\text{clean}}$ and $\mathcal{M}_{\text{adv}}$ refer to the clean, and adversarial memory bank, respectively. As for the loss function, Xu & Yang (2020) tested different memory bank and batch normalization combinations, and reported good results for a combination they refer to as ACC, indicating that the adversarial perturbation is injected into the query encoder, while the key encoder does not observe any perturbation and also the clean memory bank $\mathcal{M}_{\text{clean}}$ is used. The formulation is given in Table 2. Intuitively, the ACC loss function trains the query encoder f_q to classify adversarial inputs as its clean augmentation. To create the adversarial perturbation, Xu & Yang (2020) use PGD as well, but with the MoCo loss instead of the cross-entropy loss. Finally, the overall training loss is calculated as a weighted sum of the standard MoCo loss solely trained on transformed clean data, and the selected, e.g., ACC loss to incorporate adversarial instances.

3 METHOD

Multiple approaches for adversarial training are outlined in Section 2. Our goal is to develop a method that is not specifically tailored to one approach, but rather generalizable between different sorts of adversarial training. Therefore, given an input x and a neural network f, $f(x)$ denotes the general output vector. In supervised learning, this vector would be the logits, while in self-supervised learning, this would be the 128-dimensional output vector of the projector.

After an adversarial input x' and in consequence also its output vector $f(x')$ has been created by one of the approaches in Section 2, we create multiple pseudo adversarial inputs $f(x'_s)$ by adding normal distributed random noise,

$$f(x')_s = f(x') + \mathcal{N}(0,1) \cdot \delta_{x',x} \cdot \alpha_s \,, \tag{3}$$

where $\delta_{x',x}$ is defined as

$$\delta_{x',x} = f(x') - f(x) \tag{4}$$

and α_s is a hyperparameter, used to scale the normal distribution to reasonable values. Later on, we will show that using the same amount of perturbation during the whole training is not useful, and can even harm the performance. The parameter α_s fulfils a similar role to a learning rate step size, which is usually reduced during training. Intuitively, $\delta_{x',x}$ defines the element-wise difference that the initial adversarial input is moved away from the original instance in the decision space, while α_s scales this initial manipulation as desired.

To confirm that this type of decision space data manipulation is suitable, we create randomly perturbed adversarial instances for a standard, clean trained model (ST) and track their classification behaviour. In Figure 2, the results for an ST model are shown in the most left bar of each group. The blue (bottom) portion of the bar indicates the percentage of pseudo adversarial inputs being classified the same, as the initial adversarial input. The orange (middle) portion indicates the number of pseudo adversarial inputs returning to the classification area of the initial clean sample, and the green (top) portion gives the percentage of samples that move to a third classification area, which is neither the class of the clean nor the initial adversarial sample.

We can observe that for sufficiently small perturbation 100% of the pseudo adversarial inputs are classified the same, as the initial adversarial input. This demonstrates empirically, that the applied conditioned random noise as a form of data augmentation in the decision space can also be 'label preserving'. Only with larger perturbation radius, more and more perturbed adversarial inputs move towards a third classification area. The samples returning to their originally true class, however, can be ignored, since the adversarial instances are labelled to have the same class as their clean counterparts during training. Therefore, the assigned label for these instances would not change.

This observation is underlined by an early study of Tabacof & Valle (2016). They also found that randomly perturbing adversarial instances only affects the classification at larger amounts of pertur-

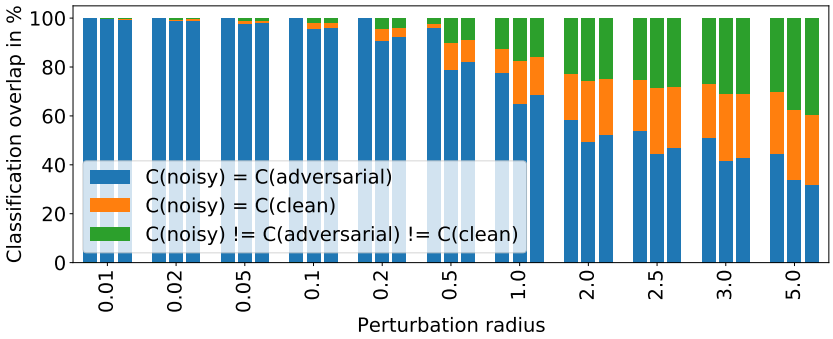


Figure 2: Percentages of pseudo adversarial inputs being classified as indicated, depending on the perturbation scaling factor α_s. The bars of each group show the results based on the following models: **Left bar:** Standard trained model; **Middle bar:** Adversarial trained model; **Right bar:** Noisy adversarial trained model. C(noisy) = C(adversarial) indicates the noisy adversarial input is classified the same, as the initial adversarial input. C(noisy) = C(clean) gives the percentage of pseudo adversarial inputs, which return to the original true classification area, while C(noisy) != C(adversarial) != C(clean) gives the percentage of pseudo adversarial inputs moving to some different, third class when perturbed randomly.

bation. However, they apply random perturbation in pixel space, which alone cannot tell us whether the corresponding output vectors of the randomly manipulated adversarial instances are close to the output vector of the initial adversarial instance.

A different perspective to the classification changes shown in Figure 2 is to empirically evaluate the local smoothness of the decision surface. If already for small random perturbation an instance moves into another classification area, the decision boundary might be sharply twisted at that point. If only at larger perturbations, the instances move into another class area, the decision boundary can be assumed to be more smooth.

The second bar of each group displays the corresponding behaviour for an adversarially trained model. Similarly to the clean model, at small perturbation radii, almost all pseudo adversarial instances are classified the same as the initial adversarial instance. However, with increasing manipulation, more and more noisy instances move to a third classification area. This also indicates, that for training with noisy adversarial instances, the scatter radius should be reduced over time. Thereby, the risk of assigning instances in a third classification area with a potentially incorrect label could be minimized.

As a final comparison, the third bar of each group shows the corresponding classifications for pseudo adversarial instances on an N-AT model. Here we can see that the number of pseudo adversarial instances being classified as the initial adversarial instance is higher compared to normal adversarial training. The number of instances moving to a third classification area is smaller as well for the N-AT model compared to the AT model. This indicates a smoother local decision boundary when a model is trained with N-AT compared to AT. Only for a very large perturbation radius, this changes in favour of AT.

Having verified that applying random perturbation as a form of data augmentation in the decision space is a valid option, our overall pseudo-code is given in Algorithm 1. Aside from the scalar for the perturbation radius α_s, we also introduce a hyperparameter to define the number of additionally created pseudo adversarial instances s_k. Each additional pseudo adversarial instance only requires the calculation of random noise and evaluation of the given loss function. Addition and multiplication to create the pseudo adversarial instances regarding time complexity are in $O(1)$, while evaluating the loss function, independent from the parameters added for N-AT, can also be considered to be in $O(1)$. Therefore, our extension to implement N-AT adds a time complexity in $O(n)$ with the number of created pseudo adversarial inputs to the overall training procedure. In Table 5 in Appendix A.3, the additional time demand for each scattered input during the different training methods, is empirically evaluated and listed.

To even out the effect of having multiple pseudo adversarial instances, we calculate the mean loss and add it, weighted by some factor λ, to calculate the overall loss as

$$\mathcal{L}_{\text{total}} = \iota\mathcal{L}_{\text{clean}} + \kappa\mathcal{L}_{\text{adv}} + \lambda\mathcal{L}_{\text{scatter}}, \tag{5}$$

where ι, κ, and λ could be different weights for the different loss functions.

Algorithm 1: Noisy Adversarial Training (N-AT).

Input: Dataset $\mathbb{D}$, model f, parameter θ, Loss function $\mathcal{L}$, # attack steps k, # scatter instances s_k, scatter scalar α_s

foreach *iter* $\in$ number of training iteration **do**
  foreach $x \in$ minibatch $B = \{x_1, \ldots, x_m\}$ **do**
    $\mathcal{L}_{\text{clean}} = \mathcal{L}\left(f\left(x\right)\right)$
    $x' = \text{generateAdversarial}\left(x\right)$
    $\mathcal{L}_{\text{adv}} = \mathcal{L}\left(f\left(x'\right)\right)$
    Noisy Adversarial Operation:
    $\delta_{x',x} = f\left(x'\right) - f\left(x\right)$
    for s_k instances **do**
      $f\left(x'\right)_s = f\left(x'\right) + \mathcal{N}\left(0,1\right) \cdot \delta_{x',x} \cdot \alpha_s$
      $\mathcal{L}_s \mathrel{+}= \mathcal{L}\left(f\left(x'\right)_s\right)$
    end
    $\mathcal{L}_{\text{scatter}} = \frac{\mathcal{L}_s}{s_k}$
    $\mathcal{L}_{\text{total}} = \iota\mathcal{L}_{\text{clean}} + \kappa\mathcal{L}_{\text{adv}} + \lambda\mathcal{L}_{\text{scatter}}$
    Optimize θ over $\mathcal{L}_{\text{total}}$
  end
end

4 Results and Discussion

Dataset and Model: All experiments were run on the Cifar-10 dataset (Krizhevsky et al. (2009)). For supervised learning, we did an additional set of experiments marked with $^+$, which uses another 1 million synthetic data points based on Cifar-10, provided by Gowal et al. (2021). The authors report an increase in adversarial robustness using the additional synthetic data. The model used for all experiments is a ResNet-18 architecture, implemented in the provided repositories of Kim et al. (2020) for RoCL, and Xu & Yang (2020) for AMOC. The experiments for the supervised case were run based on the AMOC framework. More details are provided in Appendix A.2.

Hyperparameters for training: For all experiments, we used the provided hyperparameters suggested by Kim et al. (2020) for RoCL, and Xu & Yang (2020) for AMOC, when applicable. More details are provided in Appendix A.2.

Attacks: During training, the adversarial inputs were created governed by a perturbation size of $\varepsilon = 8/255$ regarding ℓ_∞. Therefore, the ℓ_∞ attacks are referred to as seen, even if only for a small perturbation size, while the ℓ_2 and ℓ_1 attacks were completely unseen during the training procedure. For adversarial training, we used the parameters provided by the respective frameworks, listed in Appendix A.2.

To challenge the trained models, the adversarial inputs were created over 20 iteration steps, with a relative step size of 0.1 to the given allowed amount of perturbation. The overall evaluation was conducted based on the respective functions in the AMOC framework, which itself draws the attacks from the foolbox framework (Rauber et al. (2017)).

Hyperparameters for N-AT: For N-AT, we found that a good number of additional inputs is $s_k = 10$. Introducing too many additional data points would add too much noise to the training process and thereby reduces the overall performance. On the other hand, too few pseudo adversarial instances would not have any impact on the overall performance. Similarly, setting the scatter radius too small results in no effect on the results, while setting it too large, as shown in Figure 2, will move

the pseudo adversarial inputs increasingly towards and over the decision boundary of a different classification area. For supervised N-AT, we found that a surprisingly large initial $\alpha_s = 2.5$ decayed by a cosine scheduler, yields the best results. Training AMOC, setting the initial $\alpha_s = 0.25$ decayed by a cosine scheduler works best, respectively for RoCL an initial $\alpha_{0.1}$ decayed by a stepwise function reducing the initial α_s by 0.01 every 100 epochs.

For the weight of the scatter loss to the overall loss, we found that in supervised adversarial training the same weight for the original adversarial loss and the scatter loss works best. Similar to AMOC pretraining, an equal contribution of the clean, the original adversarial, and the scatter loss yields the best results. For RoCL, a weight of $\lambda = 0.25$ for the scatter loss yields the best results, combined with a weight of $\iota = \kappa = 1.0$ for the clean and original adversarial loss.

4.1 Results

A reduced version of the results for the supervised experiments is given in Table 1, where each value represents the mean value over 5 different runs. The results for all observed attacks are given in Table 6 in Appendix A.4. Table 7 in Appendix A.4 summarizes the results based on the pretrained models. The upper part reports the results where only the classification head was optimized, while the parameters of the pretrained model were frozen. The lower part, indicated by *Self-supervised + finetune*, reports the results where also the parameters of the pretrained model were optimized during training of the classification head. An **N-** in front of the given method indicates, that our proposed adaptation was applied. The results for experiments run for 200 epochs are also the mean value over 5 different runs.

Method	A_{nat}	seen			unseen			
		l_∞			l_2		l_1	
		ϵ 8/255	16/255	32/255	0.25	0.75	7.84	16.16
$\mathcal{L}_{\text{CE}}$	**93.92**	0.00	0.00	0.00	8.27	0.00	15.07	0.61
AT	81.85	52.49	22.21	1.25	**73.83**	50.91	**70.52**	54.66
N-AT	76.60	**53.05**	**26.78**	**2.36**	69.63	**52.02**	67.04	**54.95**
$\mathcal{L}_{\text{CE}}^+$	**95.04**	0.00	0.00	0.00	12.42	0.04	21.75	1.84
AT$^+$	84.15	59.22	29.70	2.60	76.78	56.09	73.47	58.05
N-AT$^+$	**84.20**	**59.80**	**30.49**	**2.81**	**76.89**	**56.86**	**73.61**	**58.81**

Table 1: Results on Cifar-10 for supervised trained models with standard cross-entropy training $\mathcal{L}_{\text{CE}}$, adversarial PGD training (AT), and our proposed Noisy Adversarial Training (N-AT). For the experiments marked with $^+$, 1 million additional synthetic data points based on Cifar-10 were used for training. During training, the initial adversarial instances were created governed by ℓ_∞ with a strength of $8/255$. All experiments were run 5 times and the mean value is reported.

4.2 Discussion

Taking a look at the results of the supervised methods in Table 1, we can reaffirm that additional synthetic data increases the clean accuracy whenever used. Also, the robust accuracy against unseen attacks increases for the clean trained model, when more input data is employed. When AT is trained on the additional data, we can also confirm that the clean, as well as robust accuracy, improves, as Gowal et al. (2021) reported. N-AT trained models, without the additional data, improve the robustness regarding the seen attacks even further, yielding a relative increase of 1.07% for small, and a relative increase of 89.26% for the largest observed perturbation. For small unseen attacks, the robustness decreases but increases for large perturbations by the unseen attacks. For the largest ℓ_2 perturbation, we can report a relative increase of 2.18%, for the largest attack based on ℓ_1 of 0.53%.

When both, additional synthetic data in the input space, as well as N-AT to artificially increase the observed adversarial instances in the decision space are used, we can not only even out the further reduction in clean accuracy observed for training without the additional data, but even increase the clean accuracy slightly, while also improving the robustness against all observed attacks.

Observing the results for AMOC when only the classification head is trained, given in Table 7 in Appendix A.4, we can report similar behaviour. The clean accuracy is slightly reduced, while the

classification accuracy for seen attacks increases relatively between 0.04% to 46%, depending on the attack size. Also the robustness against. Also, when AMOC is trained for 1000 epochs, the robust accuracy for large unseen attacks increases.

For RoCL, introducing our proposed pseudo adversarial inputs into the self-supervised pretraining, the clean accuracy relatively increases by up to 1.64%. Also, the robustness against seen attacks increases for small and medium-sized attacks. Interestingly, the robustness for large seen attacks only increases when N-RoCL during pretraining and N-AT for the classification head is applied. Similar to AMOC, RoCL also becomes more robust to medium and/or large unseen attacks, when trained with additional pseudo adversarial inputs. Particularly for the combination N-RoCL + AT, our enhanced pretraining leads to better clean accuracy and robustness against almost all attacks compared to standard RoCL + AT.

When during training of the classification head also the parameters of the pretrained models are fine-tuned, we observe an increase in clean, as well as robust accuracy for AMOC, too. In particular, for the combination N-AMOC + N-AF, compared to AMOC + N-AF, we observe that the performance increases against almost all attacks. If we assume AMOC + AF as the reported baseline, N-AMOC + N-AF relatively increase the robustness against all seen attacks between 0.47% and 17.96%, as well as against medium and large unseen attacks between 0.21% and 0.38%.

To further investigate why noisy adversarial training is sometimes weaker regarding unseen attacks, we calculated the perturbation size of successful ℓ_2 and ℓ_1 governed attacks regarding ℓ_∞. The resulting distributions are given in Figure 3 in Appendix A.5, where the x-axis indicates the perturbation size regarding ℓ_∞, and the y-axis shows the frequency of successful attacks. We recommend viewing the figures digitally to zoom in further. The distribution of manipulation sizes based on attacks controlled by ℓ_2 is given in blue (legend top), while the values for ℓ_1-attacks are shown in orange (legend middle), and for ℓ_∞-attacks in green (legend bottom). The grey vertical line gives a landmark of a perturbation of $\ell_\infty = 8/255$, which is the perturbation size seen during adversarial and noisy adversarial training. The top row of each pair shows the corresponding distributions for small perturbation size, while the bottom row shows the respective distribution for large perturbation size.

The left pair shows the results when the attacked model was trained on clean data only. We can see that the applied manipulation of attacks governed by ℓ_2 and ℓ_1 is generally lower than the adversarial manipulation applied by the corresponding ℓ_∞-attack. This could explain why even models trained on clean samples are, to some extend, robust against ℓ_2 and ℓ_1 controlled attacks.

The second and third columns show the resulting perturbation size distributions for attacks on an adversarial trained network, resp. noisy adversarial trained model. Here we can see that the perturbation of ℓ_2- and ℓ_1-attacks is larger regarding ℓ_∞ than the perturbation of the corresponding ℓ_∞-attack, especially for a small perturbation size. Since during training both models have seen adversarial samples of the perturbation size $\ell_\infty = 8/255$, this indicates why both also become more robust, but not perfect, against ℓ_2- and ℓ_1-attacks in general, but probably not why N-AT performs worse than standard AT on unseen attacks.

To further investigate the reason why N-AT might be worse regarding small perturbations by ℓ_2- and ℓ_1-attacks compared to AT, we also tracked the applied perturbation on a pixel level. The resulting perturbations, exemplary for the blue color channel of the observed input, are shown in Figure 4 in Appendix A.5 for a standard trained model, in Figure 5 for an AT trained model, and in Figure 6 for a noisy adversarial trained model. The visualization indicates whether the pixel value of the adversarial input was increased (red) or decreased (blue), regarding the pixel value of the original input.

In all cases, we observe that ℓ_2 and ℓ_1 governed attacks tend to only slightly perturb the vast majority of pixel values while selecting a handful of pixels that are heavily perturbed. This is because the overall perturbation radius for ℓ_2 and ℓ_1 is calculated over all pixels. Those attacks tend to spend their perturbation budget on the pixels, which seem to have the most impact on the classification. When the attack has the freedom to perturb each pixel independently, as is the case for ℓ_∞-attacks, the overall perturbation is larger. This also underlines the observation that clean trained models are more robust to ℓ_2- and ℓ_1-attacks while being completely defenceless against attacks controlled by ℓ_∞. Particularly, when including additional input data during training, which introduces a larger variety of pixel value combinations.

Still, the question, why N-AT is less robust against small perturbation by ℓ_2- and ℓ_1-attacks than AT, is not answered. During training, we artificially increase the number of adversarial output vectors by sampling around the initial adversarial output vector, which itself was created based on an ℓ_∞ attack. Since the initial adversarial instance was created based on a certain distribution of manipulations, which is very large for each pixel, we hypothesize that N-AT might overfit to the observed perturbations, or more precisely, the resulting adversarial output vector representations. In particular, since Cifar-10 includes only 50,000 samples.

This assumption is supported by the results for the supervised trained models, reported in Table 6 in Appendix A.4, which used the additional 1 million samples for training. This additional data seem to prevent N-AT from overfitting to the observed adversarial perturbation, as the variability in the input data, and thereby the variability in the pseudo adversarial inputs, increases. This results in higher robustness to unseen attacks, compared to standard AT. Another future step to prevent the potential overfitting would be to further investigate the manipulation distributions of ℓ_2- and ℓ_1-attacks, and in particular the distribution of their respective output vectors in the decision space. The gained insights could help to apply more sophisticated data augmentation in the decision space than the simple conditioned random noise we use here. Also, observing the distribution of clean sample output vectors could help to prevent pseudo adversarial inputs from jumping into a third classification area, as shown in Figure 2.

5 Conclusion

Using data augmentation and larger datasets have shown to be supporting and sometimes even essential (Riquelme et al. (2021)) to achieve better classification results and better generalisation. Starting from simple data augmentation methods (Krizhevsky et al. (2012)), the search for optimal data augmentation continues with modern data-driven approaches (Cubuk et al. (2020)). Another branch of research looks at how to create better and more realistic data based on a given distribution, e.g. Ho et al. (2020).

However, using these techniques does not yield robustness against adversarial manipulations. Instead, techniques like adversarial training are necessary to harden neural networks against unforeseen perturbations, which can fool the classification.

Contrary to clean inputs, which stem from the input space, adversarial inputs are created in and defined by the decision space. Therefore, we proposed to combine adversarial training with data augmentation in the decision space, referring to as Noisy Adversarial Training (N-AT), to artificially increase the observed number of adversarial instances. We show, that already applying simply conditioned random noise to the output vectors of adversarial inputs can increase the robustness, and in some cases even the clean accuracy.

Extending standard Adversarial Training (AT) (Madry et al. (2017)) to N-AT, increases the robustness against seen attacks relative to AT by 1% for small perturbations, and up to 89.26% for a larger attack radius. Also, for large unseen attacks, the robustness increases using N-AT. In particular, when using additional input data and N-AT to artificially enrich the number of observed adversarial inputs, the robustness against all observed attacks, as well as the clean accuracy is increased. Similar observations are made for self-supervised adversarial training methods. We show, that their adversarial, as well as in some cases the clean accuracy increases, extending the given method by N-AT.

On further investigation, why N-AT does not always improve the robust accuracy, in particular for small and medium unseen attacks, we hypothesize that the proposed method in its current form overfitted to the manipulation distribution introduced by the ℓ_∞-attacks used during training. Therefore, we propose to further investigate and map the decision space. Evaluating the distribution of adversarial and clean output vectors can yield insights into more suitable data augmentation methods in the decision space. These insights could increase the robustness, as well as the clean accuracy, further.

Reproducibility Statement

In order to make our results reproducible, we listed all used parameters and changes we applied for training in the Appendix A.2, and cited the corresponding papers on which our method is build

on (Kim et al. (2020); He et al. (2020)) which provide publically available github repositories themselves (https://github.com/Kim-Minseon/RoCL, https://github.com/MTandHJ/amoc). For the review process, we uploaded our code as supplementary data, and will make it publically available afterwards.

REFERENCES

Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pp. 284–293. PMLR, 2018.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pp. 39–57. IEEE, 2017.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 702–703, 2020.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Sven Gowal, Sylvestre-Alvise Rebuffi, Olivia Wiles, Florian Stimberg, Dan Calian, Timothy Mann, and London DeepMind. Doing more with less: Improving robustness using generated data. 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020.

Minseon Kim, Jihoon Tack, and Sung Ju Hwang. Adversarial self-supervised contrastive learning. *arXiv preprint arXiv:2006.07589*, 2020.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Anurag Ranjan, Joel Janai, Andreas Geiger, and Michael J Black. Attacking optical flow. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2404–2413, 2019.

Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *arXiv preprint arXiv:2106.05974*, 2021.

Pedro Tabacof and Eduardo Valle. Exploring the space of adversarial images. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 426–433. IEEE, 2016.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *31st Conference on Neural Information Processing Systems (NIPS)*, volume 30, pp. 1–18, 2017.

Andrew Tao, Karan Sapra, and Bryan Catanzaro. Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821*, 2020.

Nils Worzyk, Hendrik Kahlen, and Oliver Kramer. Physical adversarial attacks by projecting perturbations. In *International Conference on Artificial Neural Networks*, pp. 649–659. Springer, 2019.

Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. Adversarial examples improve image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 819–828, 2020.

Cong Xu and Min Yang. Adversarial momentum-contrastive pre-training. *arXiv preprint arXiv:2012.13154*, 2020.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. *arXiv preprint arXiv:2106.04560*, 2021.

A Appendix

A.1 Optimization problems and loss functions

In Table 2 the respective loss functions and optimization problems used in training AMOC and RoCL are listed, with the explanation of the used symbols in the caption.

Contrastive loss	$\mathcal{L}_{\text{con},\theta}\left(x, \{x_{\text{pos}}\}, \{x_{\text{neg}}\}\right) := -\log \frac{\sum_{\{f(x)_{\text{pos}}\}} \exp\left(\text{sim}\left(f(x), \{f(x)_{\text{pos}}\}\right)/\tau\right)}{\sum_{\{f(x)_{\text{pos}}\}} \exp\left(\text{sim}\left(f(x), \{f(x)_{\text{pos}}\}\right)/\tau\right) + \sum_{\{f(x)_{\text{neg}}\}} \exp\left(\text{sim}\left(f(x), \{f(x)_{\text{neg}}\}\right)/\tau\right)}$
RoCL	$\arg\min_{\theta} \mathbb{E}_{x\sim\mathbb{D}} \left[\max_{\delta\in B(t_1(x),\varepsilon)} \mathcal{L}_{\text{con},\theta}\left(t_1(x)+\delta, \{t_2(x)\}, \{t_1(x)_{\text{neg}}\}\right)\right]$
MoCo Loss	$\mathcal{L}_{\text{NCE}} = -\log \frac{\exp(q\cdot k_{\text{pos}}/\tau)}{\exp(q\cdot k_{\text{pos}}/\tau) + \sum_{k_{\text{neg}}\in\mathcal{M}} \exp(q\cdot k_{\text{neg}}/\tau)}$
AMOC	$\min_{\theta_q,\theta_k} \mathbb{E}_{x\in\mathbb{D}} \mathbb{E}_{t_1,t_2\in\mathcal{T}} \max_{\|\delta\|,\|\delta'\|\leq\varepsilon} \mathcal{L}\left(t_1(x)+\delta, t_2(x)+\delta', \mathcal{M}_{\text{clean}}, \mathcal{M}_{\text{adv}}\right)$
AMOC ACC	$\mathcal{L}_{\text{ACC}} = \mathcal{L}_{\text{NCE}}\left(f_q\left(t_1(x)+\delta; \text{BN}_{\text{adv}}\right), f_k\left(t_2(x); \text{BN}_{\text{clean}}\right), \mathcal{M}_{\text{clean}}\right)$

Table 2: Loss functions and optimization problems defined for the used self-supervised adversarial training methods. In all formulations x is a given clean sample, f indicates the observed model, and δ is the adversarial perturbation. Also t_1 and t_2 define two different random transformation, while τ is some temperature hyperparameter. The contrastive loss is used within the RoCL framework, based on SimCLR, where x_{pos} and x_{neg} give the positive and negative samples, respectively. The similarity sim between two output vectors is calculated as the cosine similarity. For the MoCo Loss, q and k represent the query and key encoder, respectively, and $\mathcal{M}$ is the used memory bank. While standard MoCo only defines one memory bank, within the AMOC framework the authors use two memory banks $\mathcal{M}_{\text{clean}}$ and $\mathcal{M}_{\text{adv}}$ to store the clean and adversarial historical samples. AMOC ACC is one specific loss function within the overall AMOC framework, for which the authors report good results, and which is therefore used in this study.

A.2 TRAINING PARAMETERS

In Table 3 all necessary hyperparameter for training the AMOC models are given, as well as for the supervised models/classification heads with clean, i.e., $\mathcal{L}_{\text{CE}}$, and adversarial, i.e., AT, samples. The explanation for the certain abbreviations, e.g., which transformation are used under the term simclr, are explained in the caption. The same applies for the comprehensive list of hyperparameters for training RoCL and the respective classification heads, given in Table 4.

Further details:

- All experiments were run on NVIDIA GeForce GTX 1080.
- For RoCL training, we were not able to use the suggested batch size of 256 per GPU with our hardware.
- For RoCL we changed the projector to consist of 2, instead of 1, linear layers, followed by a normalization layer.
- Finetuning RoCL with only adversarial inputs led in our experiments to a classification accuracy of 10%. Using additional clean samples, we achieved a robust accuracy around 30%, which is 10% lower than the reported values, and would not be comparable to standard adversarial training. Therefore, RoCL + AF was excluded from our experiments.

A.3 ADDITIONAL TIME DEMAND

In Table 5 the time required for one epoch of the indicated adversarial training method is listed. Further down, we split the time demand into the creation of the initial adversarial instance, which already takes up between 40.97 to 66.92% of the overall time. Calculating $\delta_{x',x}$ is only required once. Because for AT, the output vector of the clean sample is not calculated during training, the proportional time requirement is comparable high to the unsupervised methods, where the output vector is already calculated independent of our adaptation. Creating each pseudo adversarial input only adds a small portion, between0.2 to 0.52% to the overall time demand per epoch. For RoCL the evaluation of the loss function furthermore takes up 84.96% of the time to create one pseudo adversarial instance. We explain this comparable large time demand by the fact that the RoCL framework implements the loss function itself, while AMOC uses the cross entropy loss provided by pytorch and does very limited own computation in context of the loss evaluation.

A.4 ADDITIONAL RESULT FOR THE SUPERVISED THE SELF-SUPERVISED MODELS

A.5 DIFFERENCES BETWEEN THE ATTACKS

	AMOC 200	AMOC 1000	AT head	AT	$\mathcal{L}_{\text{CE}}$ head	$\mathcal{L}_{\text{CE}}$
GPU	1	1	1	1	1	1
optimizer	sgd	sgd	sgd	sgd	sgd	sgd
momentum	0.9	0.9	0.9	0.9	0.9	0.9
weight decay	5e-4	5e-4	5e-4	5e-4	2e-4	2e-4
learning rate	0.1	0.1	0.1	0.1	0.1	0.1
- decay	cosine	cosine	FC	TOTAL	FC	TOTAL
epochs	200	1000	25	40	25	40
warmup epochs	10	10				
batch size	256	256	128	128	128	128
transform	simclr	simclr	default	default	default	default
attack:						
type	ℓ_∞	ℓ_∞	ℓ_∞	ℓ_∞		
ε	8/255	8/255	8/255	8/255		
step size	2/255	2/255	2/255	2/255		
# steps	5	5	10	10		
attack weight κ	0.5	0.5	1.0	1.0		
scatter operation:						
s_k	10	10	10	10		
α_s	0.25	0.25	2.5	2.5		
scatter decay	cosine	cosine	cosine	cosine		
scatter weight λ	0.5	0.5	1.0	1.0		
MoCo specific:						
dim_mlp	512	512				
dim_head	128	128				
τ	0.2	0.2				
# samples in $\mathcal{M}_{\text{clean}}$	32768	32768				
# samples in $\mathcal{M}_{\text{adv}}$	32768	32768				
key encoder momentum	0.999	0.999				

Table 3: Full list of parameters for training AMOC, as well as the supervised models/classification heads with clean, i.e., $\mathcal{L}_{\text{CE}}$, and adversarial, i.e., AT, samples. FC is implemented to decaying the learning rate by a factor of 10 at epochs 10 and 15, while TOTAL reduces the learning rate by a factor of 10 at epochs 30 and 35. A default transformation is implemented as padding by 4, random resized cropping to 32, random horizontal flipping. SimCLR as transformation is composed of: random cropping of size 32, applying color jitter with a strength of 0.4 to the brightness, contrast, and saturation, while the hue is perturbed with strength 0.1, all with a probability of 0.8, random grayscale with a probability of 0.2, applying gaussian blur with a probability of 0.5, and random horizontal flipping. All inputs are converted to tensors, i.e., to the range [0, 1].

	RoCL	AT head	$\mathcal{L}_{\text{CE}}$ head
GPU	2	1	1
base optimizer	SGD	SGD	SGD
- momentum	0.9	0.9	0.9
- weight decay	1e-6	5e-4	5e-4
- learning rate	0.1	0.2	0.2
optimizer	LARS		
- eps	1e-8		
- trust_coeff	0.001		
learning rate decay	cosine		
warmup	GradualWarmUp		
- lr multiplier	15		
- warumup epochs	10		
epochs	1000	150	150
batch size	128 per GPU	128	128
transform	simclr	simclr	simclr
attack:			
type	ℓ_∞	ℓ_∞	
ε	0.0314 ($\approx 8/255$)	0.0314 ($\approx 8/255$)	
step size	0.007 ($\approx 2/255$)	0.007 ($\approx 2/255$)	
# steps	7	10	
attack weight κ	1.0	1.0	
scatter operation:			
s_k	10	10	
α_s	0.1	2.5	
scatter decay	stepwise	cosine	
scatter weight λ	0.25	1.0	
RoCL specific:			
τ	0.5		
λ_{RoCL}	256		

Table 4: Full list of parameters for training RoCL, as well as the supervised classification heads with clean, i.e., $\mathcal{L}_{\text{CE}}$, and adversarial, i.e., AT, samples. To train RoCL Kim et al. (2020) use the LARS You et al. (2017) optimizer based on SGD with the given parameters. The initial learning rate is increased during the first 10 epochs by an overall factor of 15. Afterwards the learning rate is decayed by a cosine scheduler. Their input transformation is composed of: applying color jitter with a strength of 0.4 to the brightness, contrast, and saturation, while the hue is perturbed with strength 0.1, all with a probability of 0.8, random grayscale with a probability of 0.2, random horizontal flipping, and random resized cropping of size 32. All inputs are converted to tensors, i.e., to the range [0, 1].

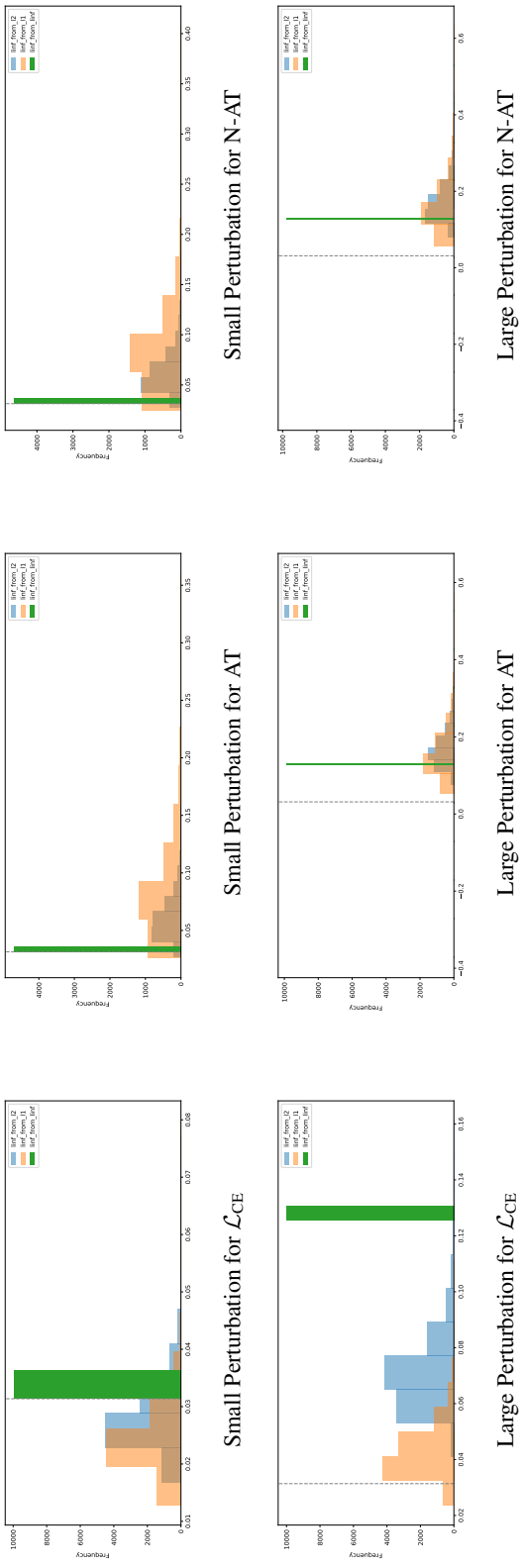
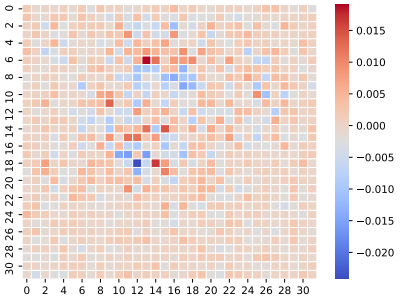
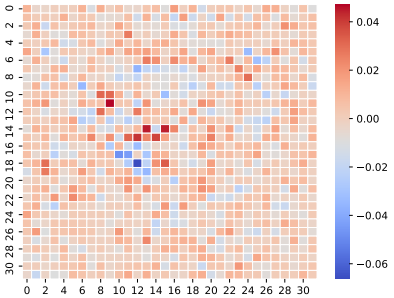


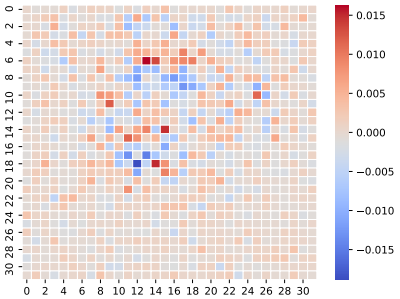
Figure 3: Distribution of perturbation size, measured regarding ℓ_∞, for attacks governed by ℓ_2 (blue), ℓ_1 (orange), and ℓ_∞ (green) on the indicated trained model. The gray line indicates a perturbation size of $\ell_\infty = 8/255$, giving a landmark for seen adversarial inputs during adversarial training.



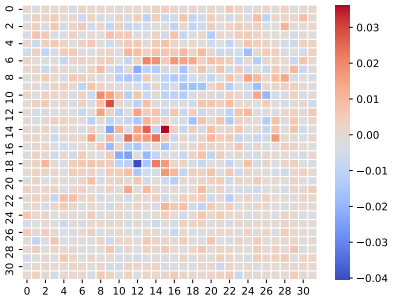
(a) Small Perturbation by ℓ_2



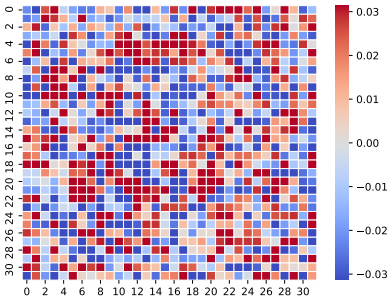
(b) Large Perturbation by $\ell 2$



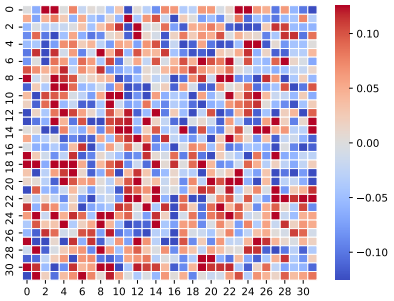
(c) Small Perturbation by ℓ_1



(d) Large Perturbation by $\ell 1$

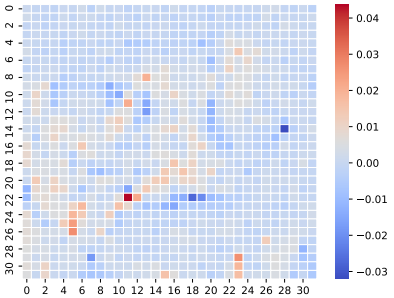


(e) Small Perturbation by ℓ_∞

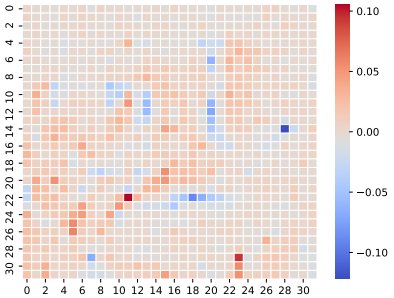


(f) Large Perturbation by $\ell\infty$

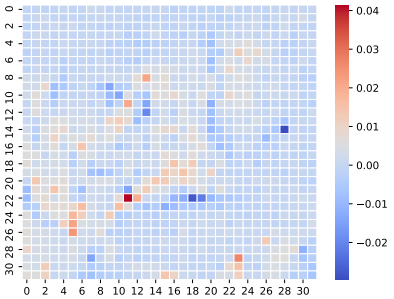
Figure 4: Perturbation for each pixel governed by ℓ_2 (top), ℓ_1 (middle), and ℓ_∞ (bottom), measured regarding ℓ_∞ on a $\mathcal{L}_{\text{CE}}$ trained model.



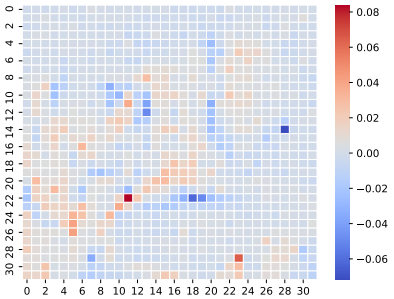
(a) Small Perturbation by ℓ_2



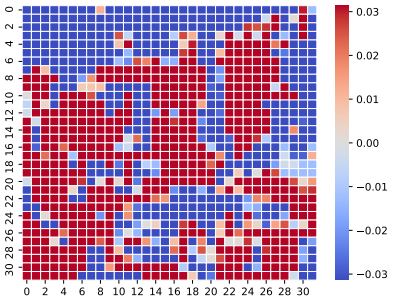
(b) Large Perturbation by $\ell 2$



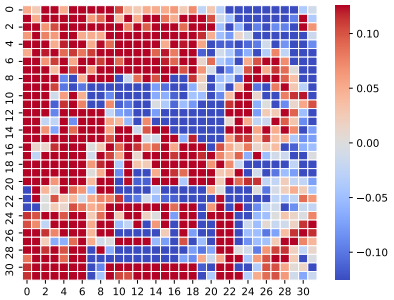
(c) Small Perturbation by ℓ_1



(d) Large Perturbation by $\ell 1$

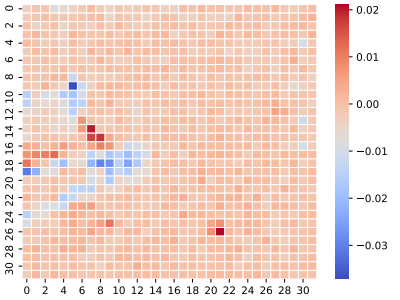


(e) Small Perturbation by ℓ_∞

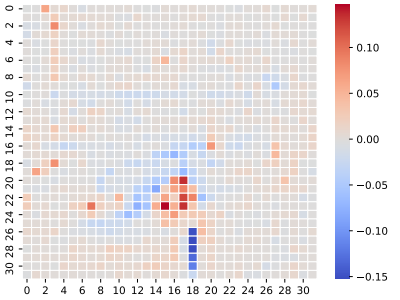


(f) Large Perturbation by $\ell\infty$

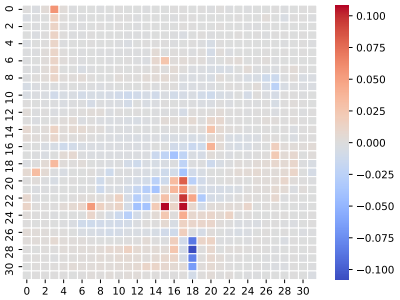
Figure 5: Perturbation for each pixel governed by ℓ_2 (top), ℓ_1 (middle), and ℓ_∞ (bottom), measured regarding ℓ_∞ on a PGD adversarial trained model.



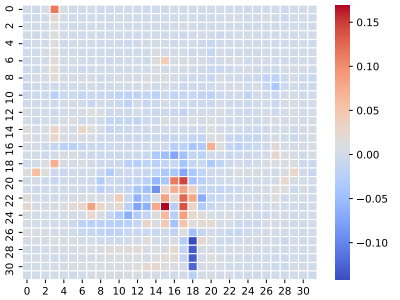
(a) Small Perturbation by ℓ_2



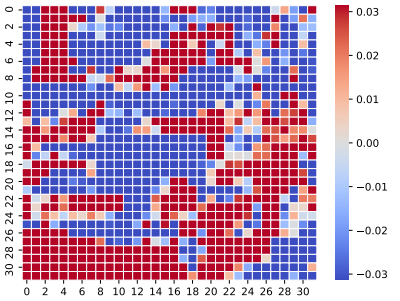
(b) Large Perturbation by $\ell 2$



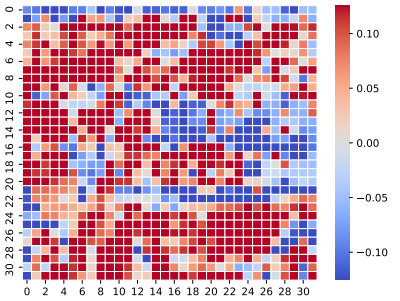
(c) Small Perturbation by ℓ_1



(d) Large Perturbation by $\ell 1$



(e) Small Perturbation by ℓ_∞



(f) Large Perturbation by $\ell\infty$

Figure 6: Perturbation for each pixel governed by ℓ_2 (top), ℓ_1 (middle), and ℓ_∞ (bottom), measured regarding ℓ_∞ on our proposed noisy adversarial trained model.

operation	mean in ms	std in ms	% of overall time
AT:			
overall time per epoch	777.55	30.37	
create initial adversarial input	429.14	16.61	55.19
calculate $\delta_{x',x}$	43.65	4.03	5.61
create one pseudo adversarial input	1.56	3.68	0.20
calculate the loss within scattering	0.09	0.02	0.01
AMOC:			
overall time per epoch	1180.50	59.58	
create initial adversarial input	483.61	26.72	40.97
calculate $\delta_{x',x}$	0.10	0.01	0.01
create one pseudo adversarial input	3.65	7.23	0.31
calculate the loss within scattering	0.26	0.19	0.02
RoCL:			
overall time per epoch	1342.06	52.27	
create initial adversarial input	898.09	35.55	66.92
calculate $\delta_{x',x}$	0.81	0.14	0.06
create one pseudo adversarial input	7.04	1.25	0.52
calculate the loss within scattering	5.98	1.22	0.45

Table 5: Time demand for different operations during N-AT given in ms. For each method we list the overall mean time and standard deviation for one epoch, as well as the time required to calculate the initial adversarial input. The overall scatter operation is split into calculating $\delta_{x',x}$, which is only performed once, and the creation of one pseudo adversarial input. In particular, we also list the time required to evaluate the loss function for the created pseudo adversarial instance.

Method	A_{nat}	seen			unseen					
		l_∞			l_2			l_1		
		ϵ 8/255	16/255	32/255	0.25	0.5	0.75	7.84	12	16.16
$\mathcal{L}_{\text{CE}}$	**93.92**	0.00	0.00	0.00	8.27	0.17	0.00	15.07	3.37	0.61
AT	81.85	52.49	22.21	1.25	**73.83**	**63.11**	50.91	**70.52**	**62.95**	54.66
N-AT	76.60	**53.05**	**26.78**	**2.36**	69.63	61.56	**52.02**	67.04	61.37	**54.95**
$\mathcal{L}_{\text{CE}}^+$	**95.04**	0.00	0.00	0.00	12.42	0.58	0.04	21.75	6.21	1.84
AT$^+$	84.15	59.22	29.70	2.60	76.78	67.51	56.09	73.47	66.06	58.05
N-AT$^+$	**84.20**	**59.80**	**30.49**	**2.81**	**76.89**	**68.08**	**56.86**	**73.61**	**66.75**	**58.81**

Table 6: Results on Cifar-10 for supervised trained models with standard cross entropy training $\mathcal{L}_{\text{CE}}$, adversarial PGD training (AT), and our proposed Noisy Adversarial Training (N-AT). For the experiments marked with $^+$, 1 million additional synthetic data points based on Cifar-10 were used for training. During training, the initial adversarial instances were created governed by ℓ_∞ with a strength of $8/255$. All experiments were run 5 times and the mean value is reported.

Method	A_{nat}	seen			unseen					
		l_∞			l_2			l_1		
		ϵ 8/255	16/255	32/255	0.25	0.5	0.75	7.84	12	16.16
Self-supervised:										
200 epochs:										
AMOC + $\mathcal{L}_{\text{CE}}$	**79.03**	36.61	7.46	0.05	**67.93**	**54.82**	**41.53**	**66.27**	**58.53**	**50.74**
N-AMOC + $\mathcal{L}_{\text{CE}}$	78.88	**37.09**	**8.15**	0.05	67.64	54.58	41.37	65.77	57.91	50.19
AMOC + AT	**74.79**	43.97	14.53	0.19	**67.10**	**58.10**	48.09	**66.03**	**60.78**	**54.92**
N-AMOC + AT	74.58	**44.57**	**15.45**	**0.26**	66.88	57.93	**48.21**	65.72	60.43	54.64
AMOC + **N-AT**	**74.32**	44.08	15.15	0.24	66.63	**57.92**	**48.21**	**65.62**	**60.78**	**54.82**
N-AMOC + N-AT	74.25	**44.59**	**15.85**	**0.28**	**66.64**	57.75	48.18	65.49	60.21	54.44
1000 epochs:										
AMOC + $\mathcal{L}_{\text{CE}}$	**86.52**	44.91	11.46	0.11	**77.04**	63.59	50.39	**75.47**	68.27	59.75
N-AMOC + $\mathcal{L}_{\text{CE}}$	85.90	**45.17**	**12.02**	**0.14**	76.78	**64.29**	**50.99**	75.38	**68.64**	**60.97**
AMOC + AT	**84.48**	50.87	16.85	0.26	**77.07**	**67.28**	56.00	**76.14**	**70.45**	**64.43**
N-AMOC + AT	83.80	**50.89**	**17.81**	**0.38**	76.35	66.79	**56.16**	75.44	69.78	64.29
AMOC + **N-AT**	**83.88**	51.00	17.46	0.33	**76.44**	66.41	55.77	**75.51**	**69.87**	63.67
N-AMOC + N-AT	83.40	**51.08**	**18.47**	**0.37**	75.97	**66.45**	**56.11**	75.15	69.48	**63.83**
RoCL + $\mathcal{L}_{\text{CE}}$	83.69	38.49	8.73	**0.66**	**65.98**	61.12	44.47	**68.03**	**67.59**	60.42
N-RoCL + $\mathcal{L}_{\text{CE}}$	**85.06**	**40.44**	**9.54**	0.63	65.37	**62.86**	**47.42**	66.42	66.63	**63.67**
RoCL + AT	79.65	47.35	16.33	**0.36**	67.33	65.18	53.38	68.20	68.17	65.58
N-RoCL + AT	**79.69**	**49.36**	**17.41**	0.33	**67.58**	**66.15**	**54.64**	**68.21**	**68.54**	**66.68**
RoCL + **N-AT**	78.63	47.31	16.29	0.25	**68.34**	64.92	53.04	**68.92**	**69.27**	65.34
N-RoCL + N-AT	**79.69**	**49.33**	**17.22**	**0.38**	67.59	**66.03**	**54.47**	68.38	68.43	**66.73**
Self-supervised *+ finetune*										
200 epochs:										
AMOC + AF	82.87	52.60	**22.11**	1.11	74.65	63.56	50.77	71.20	63.32	**54.81**
N-AMOC + AF	**83.29**	**52.98**	21.69	**1.14**	**74.84**	**63.80**	**50.96**	**71.28**	**63.33**	54.73
AMOC + **N-AF**	82.19	52.73	22.23	**1.28**	73.71	63.51	**51.04**	70.43	63.05	**54.62**
N-AMOC + N-AF	**82.60**	**52.98**	**22.34**	1.26	**74.28**	63.51	50.92	**70.81**	63.05	54.40
1000 epochs:										
AMOC + AF	83.28	52.82	**22.04**	**1.12**	74.95	63.87	**51.38**	71.79	63.83	55.13
N-AMOC + AF	**84.00**	**53.08**	21.74	1.09	**75.44**	**64.65**	51.20	**71.95**	**64.20**	**55.33**
AMOC + **N-AF**	81.85	52.62	**22.51**	**1.38**	73.77	63.21	50.99	70.82	63.16	54.55
N-AMOC + N-AF	**82.76**	**53.07**	22.17	1.32	**74.63**	**64.11**	**51.49**	**71.17**	**63.83**	**55.30**

Table 7: Results on Cifar-10 for self-supervised trained models. In the first part, the classification head was trained without adapting the pretrained features. In the second part, the parameters of the pretrained model were also adapted during training the classification head. $\mathcal{L}_{\text{CE}}$, AT, and N-AT define, whether the classification head, and in case of finetuning the pretrained models, were trained on clean, adversarial, or with addition of pseudo adversarial inputs, respectively. An **N-** before the given self-supervised method indicates, that our proposed extension was applied. During training, the initial adversarial instances were created governed by ℓ_∞ with a strength of $8/255$.