Supplementary Material for
Relating Adversarially Robust Generalization to Flat Minima

David Stutz[1] Matthias Hein[2] Bernt Schiele[1]
[1]Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken
[2]University of Tübingen, Tübingen
{david.stutz,schiele}@mpi-inf.mpg.de, matthias.hein@uni-tuebingen.de

A. Overview

In the main paper, we empirically studied the connection between adversarial robustness (in terms of the *robust* loss RLoss, i.e., the cross-entropy loss on adversarial examples) and flatness of the RLoss landscape w.r.t. changes in the weight space. In this context, we also consider the phenomenon of robust overfitting [75], i.e., that robustness on training examples increases consistently throughout training while robustness on test examples eventually *decreases*. Based on average- and worst-case metrics of flatness in RLoss, which we ensure to be scale-invariant, we show a **clear relationship between adversarial robustness and flatness**. This takes into account many popular variants of adversarial training (AT), i.e., training on adversarial examples: TRADES [102], MART [92], AT-AWP [96] AT with self-supervision [40] or additional unlabeled examples [11]. All of them improve adversarial robustness *and* flatness. Vice versa, approaches known to improve flatness, e.g., Entropy-SGD [12], weight clipping [85] or weight averaging [33] also improve adversarial robustness. Finally, we found that even simple regularization schemes, e.g., AutoAugment [21], weight decay or label noise, also improve robustness by finding flatter minima.

A.1. Contents

This supplementary material is organized as follows:

- Sec. B: additional discussion of **related work**.
- Sec. C: details on **RLoss landscape visualization** and comparison to [54] (cf. Fig. A and B).
- Sec. D: details on how to **compute our average- and worst-case flatness measures**, including ablation studies in Sec. D.1 (cf. Fig. C, D and E).
- Sec. E: discussion of **scale-invariance** of our visualization and flatness measures (cf. Fig. F and Tab. A).
- Sec. F: specifics on our **experimental setup** (training and evaluation details).
- Sec. G: discussion of all individual **methods**, including ablation regarding hyper-parameters in Sec. G.1 (cf. Fig. G and H) and flatness in Sec. G.2 (cf. Fig. I and J). Training curves for all methods in Fig. K.
- Sec. H: all **results in tabular form** (Tab. B, C, E, D)

B. Related Work

Adversarial Examples and Defenses: Adversarial examples, first reported in [89], can be generated using a wide-range of white-box attacks [89, 31, 50, 68, 63, 59, 9, 25, 57], with full access to the network, or black-box attacks [13, 6, 87, 41, 77, 64], with limited access to model queries. Besides certified and provable defenses [16, 98, 49, 101, 100, 94, 32, 30, 61, 81, 53, 17], adversarial training (AT) has become the de-facto standard, as discussed in the main paper. However, there are also many detection/rejection approaches [34, 29, 55, 58, 2, 60], so-called manifold-projection methods [42, 70, 78, 79], several methods based on pre-processing, quantization and/or dimensionality reduction [7, 71, 5], methods based on randomness, regularization or adapted architectures [99, 5, 65, 80, 38, 45, 76, 47, 52, 97] or ensemble methods [56, 84, 36, 91], to name a few directions. However, often these defenses can be broken by considering adaptive attacks [8, 10, 3, 4].

Weight Robustness: Flatness, w.r.t. the clean or robust loss surface, is also related to robustness in the weights. However, only few works explicitly study this "weight robustness": [93] considers robustness w.r.t. L_∞ weight perturbations, while [14] studies Gaussian noise on weights. [74, 37], in contrast, adversarially flip bits in (quantized) weights to reduce performance. Recently, [85] shows that robustness in weights can improve energy efficiency of neural network accelerators (i.e., specialized hardware for inference). This type of weight robustness is also relevant for some backdooring attacks that explicitly manipulate weights [46, 27]. Fault tolerance is also a related concept, as it often involved changes in units or weights. It has been studied in early works [66, 15, 22], obtaining fault tolerance

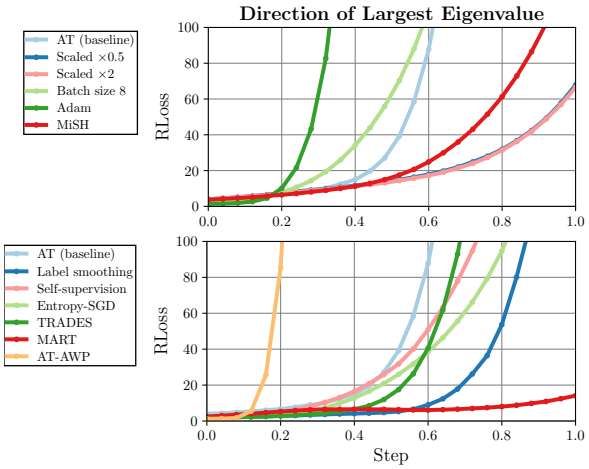


Figure A: **Visualization in "Hessian" Direction:** RLoss visualized in the direction of the largest Hessian eigenvalue (i.e., the corresponding eigenvector). The eigenvalues quantify the "rate of change" along the corresponding eigenvector. Thus, the largest eigenvalue represents a worst-case direction in weight space. Clearly, RLoss increases significantly in these directions.

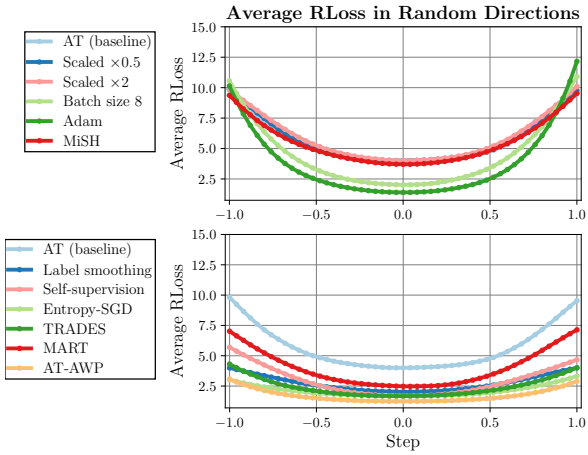


Figure B: **Filter-Wise Normalization:** Compared to the RLoss landscape visualizations in the main paper, using per-layer normalization in Eq. (2), we follow [54] and use filter-wise normalization in Eq. (1). Again, we plot mean RLoss across 10 random directions. However, this does not change results significantly, flatness remains difficult to judge and compare in an objective way. Filter-wise normalization, however, "looks" generally flatter.

using approaches similar to adversarial training NNs using approaches similar to adversarial training. However, there are also more recent works, e.g., weight dropping regularization [73] or GAN-based training [26]. We refer to [90] for a comprehensive survey.

C. Visualization Details and Discussion

Visualization Details: For the plots in the main paper, we compute the mean RLoss across 10 random, normalized directions; for adversarial directions, we plot max RLoss over 10 adversarial directions. After normalization, we rescale the weight directions to have length 0.5 for random directions and 0.025 for adversarial directions. This essentially "zooms in" and is particularly important when visualizing along adversarial weight directions. In all cases, we estimate RLoss on one batch of 128 test examples for 51 evenly spaced step sizes in $[-1, 1]$. We found that using more test examples does not change the RLoss landscape significantly. Fig. A shows additional visualizations along the direction of the largest Hessian eigenvalue (also using per-layer normalization, multiplied by 0.5).

Discussion of [54]: Originally, [54] uses a per-filter normalization instead of our per-layer normalization. Specifically, this means

$$\hat{\nu}^{(l,i)} = \frac{\nu^{(l)}}{\|\nu^{(l,i)}\|_2} \|w^{(l,i)}\|_2 \quad \text{for layer } l, \text{ filter } i, \tag{1}$$

instead of our normalization outlined in the main paper:

$$\hat{\nu}^{(i)} = \frac{\nu^{(l)}}{\|\nu^{(l)}\|_2} \|w^{(l)}\|_2 \quad \text{for layer } l. \tag{2}$$

Furthermore, [54] does not consider changes in the biases or batch normalization parameters. Instead, we also normalize the biases as above and take them into account for visualization (but not the batch normalization parameters). More importantly, [54] considers only (clean) Loss, while we focus on RLoss. Compared to the plots from the main paper, Fig. B shows that the difference between filter-wise and layer-wise normalization has little impact in visually judging flatness. Generally, filter-wise normalization makes the RLoss landscape "look" flatter. However, this is mainly because the absolute step size, i.e., $\|\hat{\nu}\|_2$, is smaller compared to layer-wise normalization: for our AT baseline, this is (on average) $\|\hat{\nu}\|_2 \approx 33.13$ for layer-wise and $\|\hat{\nu}\|_2 \approx 21.49$ for filter-wise normalization.

D. Computing Flatness in RLoss

Average-Case Flatness: The average-case flatness measure in RLoss is defined as:

$$\mathbb{E}_\nu[\max_{\|\delta\|_\infty \leq \epsilon} \mathcal{L}(f(x+\delta, w+\nu), y)] - \max_{\|\delta\|_\infty \leq \epsilon} \mathcal{L}(f(x+\delta; w), y) \tag{3}$$

where $\mathbb{E}_\nu$ denotes the expectation over random weight perturbations $\nu \in B_\xi(w)$, $\mathcal{L}$ is the cross-entropy loss and $\max_{\|\delta\|_\infty \leq \epsilon} \mathcal{L}(f(x+\delta; w), y)$ represents the robust loss

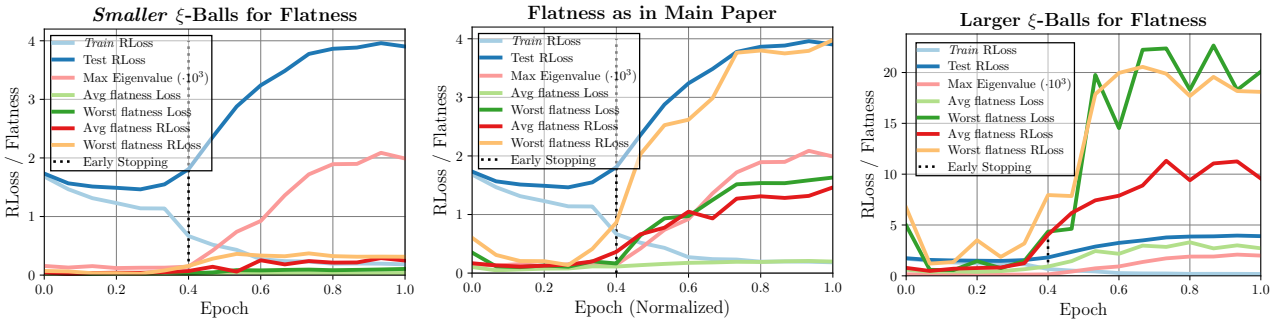


Figure C: **Flatness Throughout Training, Ablation:** We plot train and test RLoss, maximum Hessian eigenvalue $\lambda_{\max}$, average-/worst-case flatness of (clean) Loss as well as average-/worst-case flatness on RLoss. We consider ξ=0.25/ξ=0.001 (left), ξ=0.5/ξ=0.003 (middle and main paper), and ξ=0.75/ξ=0.005 for average-/worst-case flatness, respectively. If the neighborhood $b_\xi(w)$ is chosen too small (left), increases/changes in flatness during robust overfitting are difficult to measure due to fluctuations throughout training. Chosen too large (right), in contrast, worst-case flatness (both in Loss and RLoss) quickly reaches unreasonably high loss values. This becomes problematic when comparing across models.

(RLoss). The first term is computed by randomly sampling 10 weight perturbations from

$$B_\xi(w) = \{w + \nu : \|\nu^{(l)}\|_2 \leq \xi \|w^{(l)}\|_2 \forall \text{ layers } l\}. \quad (4)$$

For each weight perturbation ν, the robust loss, defined as $\max_{\|\delta\|_\infty \leq \epsilon} \mathcal{L}(f(x+\delta, w+\nu), y)$, is estimated using PGD with 20 iterations ($\epsilon = 8/255$, learning rate 0.007 and signed gradient). This is done *per-batch* (of size 128) for the *first* 1000 test examples. Alternatively, the weights perturbations ν could also be fixed across batches (i.e., 10 samples in total for $\lceil 1000/128 \rceil$ batches). However, this is not possible for our worst-case flatness measure, as discussed next. Thus, for comparability, we sample random weight perturbations *for each* batch individually. The second term is computed using PGD-20 with 10 restarts, choosing the worst-case adversarial examples per test example (i.e., maximizing RLoss).

Sampling in $B_\xi(w)$ is accomplished by sampling individually per layer. That is, for each layer l, we compute $\xi' := \xi \cdot \|w^{(l)}\|_2$ given the original weights w. Then, a random vector $\nu^{(l)}$ with $\|\nu^{(l)}\|_2 \leq \xi'$ is sampled. This is done for each layer, handling weights and biases as separate layers, but ignoring batch normalization [43] parameters.

Worst-Case Flatness: Worst-case flatness is defined as:

$$\max_{\nu \in B_\xi(w)} \max_{\|\delta\|_\infty \leq \epsilon} \mathcal{L}(f(x+\delta, w+\nu), y) - \max_{\|\delta\|_\infty \leq \epsilon} \mathcal{L}(f(x+\delta; w), y). \quad (5)$$

Here, the expectation over ν in Eq. (3) is replaced by a maximum over $\nu \in B_\xi(w)$, considering smaller ξ. In practice, the first term in Eq. (5) is computed by *jointly* optimizing over weight perturbation ν and input perturbation(s) δ on a per-batch basis (of size $B = 128$). This means, after random initialization of δ_b, $\forall b = 1, \ldots, B$, and $\nu \in B_\xi(w)$, each iteration computes and applies updates

$$\Delta_\nu = \nabla_\nu \sum_{b=1}^{B} \mathcal{L}(f(x_b + \delta_b; w + \nu), y_b) \quad (6)$$

$$\Delta_{\delta_b} = \nabla_{\delta_b} \sum_{b=1}^{B} \mathcal{L}(f(x_b + \delta_b; w + \nu), y_b) \quad (7)$$

before projecting δ_b and ν onto the constraints $\|\delta_b\|_\infty \leq \epsilon$ and $\|\nu^{(l)}\|_2 \leq \xi \|w^{(l)}|_2$. The latter projection is applied in a per-layer basis, similar to sampling as described above. For the adversarial weight perturbation ν, we use learning rate 0.001, after normalizing the update Δ_ν per-layer as in Eq. (2). We run 20 iterations with 10 restarts for each batch.

Flatness of Clean Loss Landscape: We can also consider both Eq. (3) and Eq. (5) on the *clean* (cross-entropy) loss ("Loss"), i.e., $\mathcal{L}(f(x, w+\nu), y)$ instead of $\max_{\|\delta\|_\infty \leq \epsilon} \mathcal{L}(f(x+\delta, w+\nu), y)$. We note that RLoss is an upper bound of (clean) Loss. Thus, flatness in RLoss and Loss are connected. However, Pearson correlation between RLoss and average-case flatness in (clean) Loss is only 0.27, compared to 0.85 for average-case flatness in *RLoss*. This indicates that correctly measuring flatness in RLoss is crucial to empirically establish a relationship between robustness and flatness.

D.1. Ablation for Flatness Measures

Flatness Throughout Training: Fig. C shows average- and worst-case flatness on both clean as well as robust loss (Loss and RLoss) throughout training of our AT baseline. We consider different sizes of the neighborhood $B_\xi(w)$ for computing our flatness measures: ξ=0.25/ξ=0.001 (left), ξ=0.5/ξ=0.003 (middle, as in main paper), and ξ=0.75/ξ=0.005 for average-/worst-case flatness, respectively. While average-case flatness of *clean* Loss does *not* mirror robust overfitting very well, its worst-case pendant increases during overfitting, even though RLoss is *not* taken

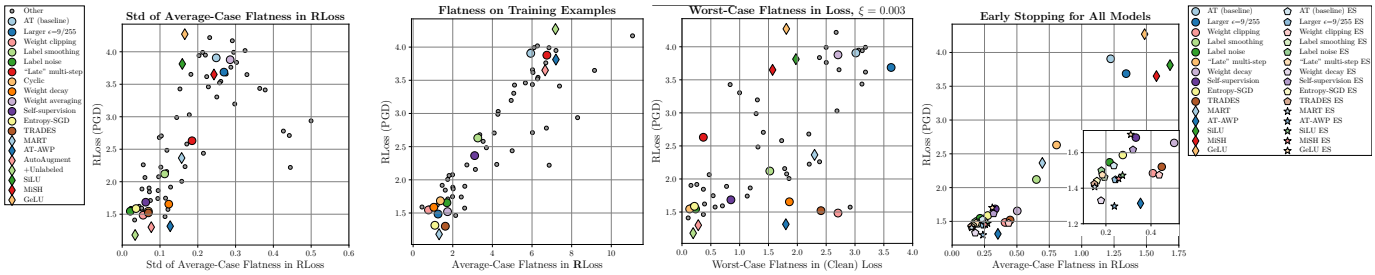


Figure D: **Left: Standard Deviation of Average-Case Flatness:** We plot RLoss (y-axis) against the standard deviation (std) in our average-case flatness measure (x-axis). Note that the standard-deviation is due to the random weight perturbations ν in Eq. (3). Interestingly, more robust methods are not only flatter, but our average-case flatness measure also has lower standard deviation. **Middle Left: Average-Case Flatness of Train RLoss:** *Test* RLoss plotted against our average-case flatness measure as computed on *training* examples. Even on the training set, flatness is predictive of robust generalization, i.e., adversarial robustness on the test set. The relationship, however, is weaker compared to average-case flatness on test examples. **Middle Right: Worst-Case Flatness in (Clean) Loss:** As worst-case flatness in the *clean* Loss landscape also mirrors robust overfitting in Fig. C, we plot RLoss against worst-case flatness in Loss. Even though flatness is measured considering clean Loss, many methods improving robustness (i.e., lower RLoss) exhibit surprisingly good flatness. **Right: Early Stopping for all Models:** RLoss vs. average-case flatness for all models where early stopping improves adversarial robustness. For example, this is not the case for AutoAugment or AT with unlabeled examples. Across all models, early stopping improves both robustness and flatness. For clarity we provide a zoomed-in plot for the lower left corner.

into account. Furthermore, if the neighborhood $B_\xi(w)$ is chosen too small, the flatness measures are not sensitive enough to be discriminative (cf. left). Fig. C also shows that, throughout training of one model, the largest Hessian eigenvalue mirrors robust overfitting. Overall, this means that early stopping essentially improves adversarial robustness by finding flatter minima. This is confirmed in Fig. D (right), showing that early stopping consistently improves robustness and flatness.

Standard Deviation in Average-Case Flatness: In Fig. D (left), the x-axis plots the standard deviation in our average-case flatness measure (in RLoss). Note that the standard deviation originates in the random samples ν used to calculate Eq. (3). First of all, standard deviation tends to be small (i.e., ≤ 0.3) across almost all models. This means that our findings in the main paper, i.e., the strong correlation between flatness and RLoss, is supported by low standard deviation. More importantly, the standard deviation *reduces* for particularly robust methods.

Average-Case Flatness on Training Examples: Fig. D (middle left) shows that average-case flatness in RLoss is also predictive for robust generalization when computed on *training* examples. However, the correlation between (test) RLoss and (train) flatness is less clear, i.e., there is a larger "spread" across methods. Here, we use the first 1000 training examples to compute average-case flatness.

Worst-Case Flatness on *Clean* Loss: In Fig. C, worst-case flatness on clean Loss also correlates with robust overfitting. Thus, in Fig. D (middle right), we plot RLoss against worst-case flatness of Loss, showing that there is no clear relationship across models. Nevertheless, many methods improving adversarial robustness also result in flatter minima in the clean loss landscape. This is sensible as RLoss is generally an upper bound for (clean) Loss. On the other hand, flatness in Loss is *not* discriminative enough to clearly distinguish between robust and less robust models.

Ablation for $B_\xi(w)$: For computing our average- and worst-case flatness measures (in RLoss), we considered various sizes of neighborhoods in weight space, i.e. $B_\xi(w)$ from Eq. (4) for different ξ. Fig. E considers $\xi \in \{0.25, 0.5, 0.75, 1\}$ for average-case flatness (top) and $\xi \in \{0.00075, 0.001, 0.003, 0.005\}$ for worst-case flatness (bottom). In both cases, we plot RLoss (y-axis) against flatness in RLoss (y-axis), as known from the main paper. Average-case flatness using small $\xi = 0.25$ results in significantly smaller values, between 0 and 0.4, i.e., the increase in RLoss in random weight directions is rather small. Still, the relationship between adversarial robustness and flatness is clearly visible. The same holds for larger $\xi \in \{0.75, 1\}$. Worst-case flatness generally gives a less clear picture regarding the relationship between robustness and flatness. Additionally, for larger $\xi \in \{0.003, 0.005\}$, variance seems to increase such that this relationship becomes less pronounced. In contrast to average-case flatness, the variance is not induced by the 10 restarts used for Eq. (5), but caused by training itself. Indeed, re-training our AT baseline leads to a worst-case flatness in RLoss of 5.1, a significant reduction from 6.49 as obtained for our original baseline. Overall, however, the observations from the main paper can be confirmed using different sizes of the neighborhood $B_\xi(w)$.

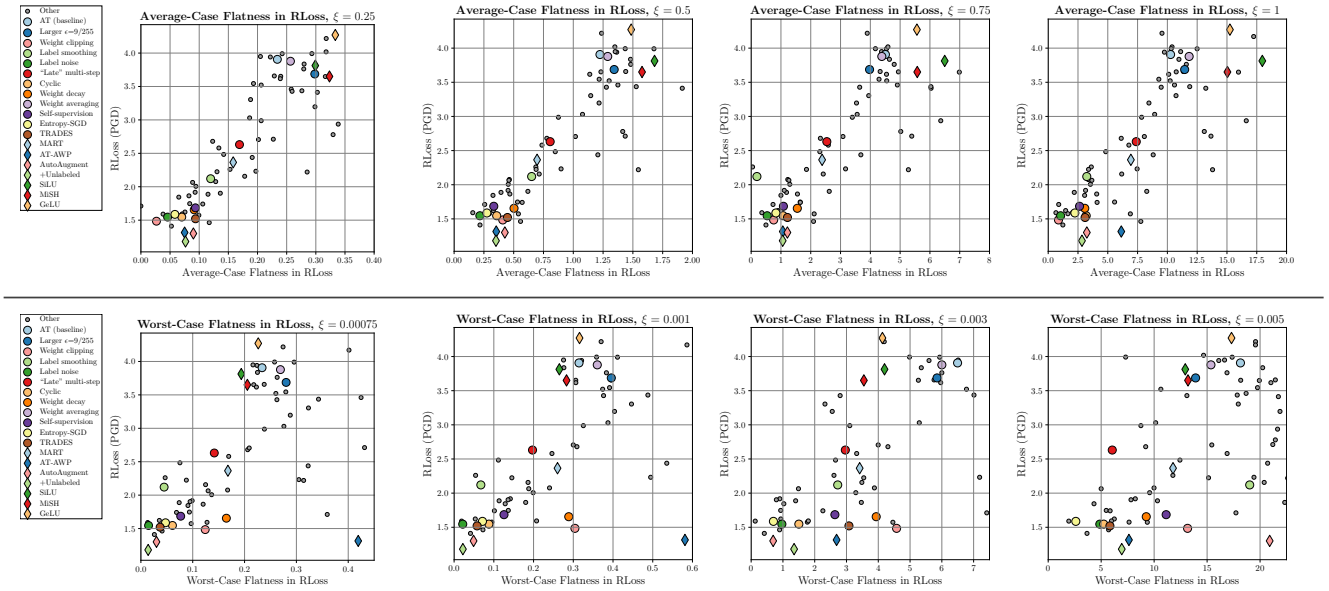


Figure E: **Flatness in RLoss, Ablation for $B_\xi(w)$:** RLoss (y-axis) plotted against average-case (top) and worst-case (bottom) flatness in RLoss (x-axis). **Top:** We consider $\xi \in \{0.25, 0.5, 0.75, 1\}$ for average-case flatness. The clear relationship between adversarial robustness, i.e., low RLoss, and flatness shown for $\xi = 0.5$ in the main paper can be reproduced for all cases. **Bottom:** For worst-case flatness, we consider $\xi \in \{0.00075, 0.001, 0.003, 0.005\}$. When chosen too large, e.g., $\xi = 0.005$, however, variance seems to increase, making the relationship less clear. For small ξ, e.g., $\xi = 0.00075$, the correlation between robustness and flatness is pronounced, except for a few outliers, including AT-AWP [96].

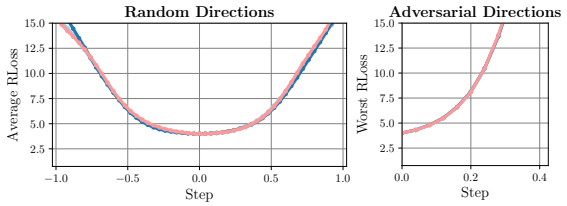
E. Scaling Networks and Scale-Invariance

Scale-Invariance: In the main paper, we presented a simple experiment to show that our measures of flatness in RLoss are scale-invariant: we scaled weights *and* biases of *all* convolutional layers in our adversarially trained ResNet-18 [35] by factor $s \in \{0.5, 2\}$. Note that all convolutional layers in the ResNet are followed by batch normalization layers [43]. Thus, the effect of scaling is essentially "canceled out", i.e., these convolutional layers are scale-invariant. Thus, the prediction stays roughly constant. Fig. F (left) shows RLoss landscape visualizations for AT and its scaled variants in random and adversarial weight directions. Clearly, scaling AT has negligible impact on the RLoss landscape in both cases. Fig. F (right) shows that our flatness measures remain invariant, as well. As $B_\xi(w)$ in Eq. (4) is defined *per-layer* (weights and biases separately) and *relative* to w, the neighborhood increases alongside the weights, rendering visualization and flatness measures invariant. When, for example, scaling up specific layers and scaling down others, as discussed in [24], causes the neighborhood $B_\xi(w)$ to increase or decrease in size for these particular layers. Thus, following [24], scaling up the first layer of a two-layer ReLU network by α and scaling down the second layer by $1/\alpha$ (keeping the output constant), has no effect in terms of measuring flatness as the per-layer neighborhood $B_\xi(w)$ is scaled accordingly, as well. The Hessian eigenspectrum, in contrast, scales with the models, cf. Tab. A, and is not suited to quantify flatness.

Convexity and Flatness: Tab. A also presents the convexity metric introduced in [54]: $|\lambda_{\text{min}}|/|\lambda_{\text{max}}|$ with $\lambda_{\text{min/max}}$ being largest/smallest Hessian eigenvalue. Note that the Hessian is computed following [54] w.r.t. to the *clean* (cross-entropy) loss, not taking into account adversarial examples. The intuition is that negative eigenvalues with large absolute value correspond to non-convex directions in weight space. If these eigenvalues are large in relation to the positive eigenvalues, there is assumed to be significant non-convexity "around" the found minimum. Tab. A shows that this fraction is usually very small, as also found in [54]. However, Tab. A also shows that this convexity measure is not clearly correlated with adversarial robustness.

F. Detailed Experimental Setup

We focus our experiments on CIFAR10 [48], consisting of 50k training examples and 10k test examples of size 32×32 (in color) and $K = 10$ class labels. We use all training examples during training, but withhold the *last* 500 test examples for early stopping. Evaluation is performed on the *first* 1000 test examples, due to long runtimes of AutoAttack [19] and our flatness measures (on RLoss). Any evaluation on the training set is performed on the first 1000 training examples (e.g., in Fig. D, middle).



Model	Robustness		Flatness		
	RErr ↓ (test)	RErr ↓ (train)	Worst *Loss*	Avg **RLoss**	Worst **RLoss**
Scaled $\times 0.5$	60.9	8.4 (-52.5)	0.86	1.36	6.50
AT (baseline)	61.0	8.4 (-52.6)	0.86	1.21	6.48
Scaled $\times 2$	61.0	8.3 (-52.7)	0.86	1.27	6.49

Figure F: **Flatness and Scale-Invariance. Left:** We plot average RLoss and worst RLoss along random and adversarial directions, as discussed in Sec. C, for AT and its scaled variants, $\times 0.5$ and $\times 2$. Clearly, RLoss landscape looks nearly identical. **Right:** Robustness against PGD-20 on train and test examples, as well as average- and worst-case flatness measures on RLoss. For completeness, we also include worst-case flatness on clean Loss. All of these measures are nearly invariant to scaling. The shown differences can be attributed to randomness in computing these measures.

As network architecture, we use ResNet-18 [35] with batch normalization [43] and ReLU activations. Our AT baseline (i.e., default model) is trained using SGD for 150 epochs, batch size 128, learning rate 0.05, reduced by factor 0.1 at 60, 90 and 120 epochs, weight decay 0.005 and momentum 0.9. We save snapshots every 5 epochs to perform early stopping, but do *not* use early stopping by default. We whiten input examples by subtracting the (per-channel) mean and dividing by standard deviation. We use standard data augmentation, considering random flips and cropping (by up to 4 pixels per side). By default, we use 7 iterations PGD, with learning rate 0.007, signed gradient and $\epsilon = 8/255$ to compute L_∞ adversarial examples. Note that no momentum [25] or backtracking [86] is used for PGD. The training curves in Fig. K correspond to robustness measured using the 7-iterations PGD attack used for training, which we also use for early stopping (with 5 random restarts).

For evaluation, we run PGD for 20 iterations and 10 random restarts, taking the worst-case adversarial example per test example [86]. Our results considering robust loss (RLoss) are based on PGD, while we report robust test error (RErr) using AutoAttack [19]. Note that AutoAttack does *not* maximize cross-entropy loss as it stops when adversarial examples are found. Thus, it is not suitable to estimate RLoss. Robust test error is calculated as the fraction of test examples that are either mis-classified or successfully attacked. The distinction between PGD-20 and AutoAttack is important as AutoAttack does *not* maximize cross-entropy loss, resulting in an under-estimation of RLoss, while PGD-20 generally underestimates RErr. Computation of our average- and worst-case flatness measure is detailed in Sec. D.

Everything is implemented in PyTorch [69].

G. Methods

In the following, we briefly elaborate on the individual methods considered in our experiments.

Learning Rate Schedules: Besides our default, multi-step learning rate schedule (learning rate 0.05, reduced by factor 0.1 after epochs 60, 90, and 120), we followed [67] and implemented the following learning rate schedules: First, simply using a constant learning rate of 0.05. Second, only two "late" learning rate reductions at epochs 140 and 145, as done in [72]. Third, using a cyclic learning rate, interpolating between a learning rate of 0.2 and 0 for 30 epochs per cycle, as, e.g., done in [95]. We consider training for up to 4 cycles (= 120 epochs). These learning rate schedules are available as part of PyTorch [69].

Label Smoothing: In [88], label smoothing is introduced as regularization to improve (clean) generalization by *not* enforcing one-hot labels in the cross-entropy loss. Instead, for label y and $K = 10$ classes, a target distribution $p \in [0,1]^K$ (subject to $\sum_i p_i = 1$) with $p_y = 1 - \tau$ (correct label) and $p_i = \tau/K-1$ for $i \neq y$ (all other labels) is enforced. During AT, we only apply label smoothing for the weight update, not for PGD. We consider $\tau \in \{0.1, 0.2, 0.3\}$.

Label Noise: Instead of explicitly enforcing a "smoothed" target distribution, we also consider injecting label noise during training. In each batch, we sample random labels for a fraction of τ of the examples. Note that the labels are sampled uniformly across all $K = 10$ classes. Thus, in expectation, the enforced target distribution is $p_y = 1 - \tau + \tau/K$ and $p_i = \tau - \tau/K / K-1$. As result, this is equivalent to label smoothing with $\tau = \tau - \tau/K$. In contrast to label smoothing, this distribution is not enforced explicitly in the cross-entropy loss. As above, adversarial examples are computed against the true labels (without label noise) and label noise is injected for the weight update. We consider $\tau \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. While label smoothing does not further improve adversarial robustness for $\tau > 0.3$, label noise proved very effective in avoiding robust overfitting, which is why we also consider $\tau = 0.4$ or 0.5.

Weight Averaging: To implement weight averaging [44], we follow [33] and keep a "running" average $\bar{w}$ of the model's weights throughout training, updated in each iteration t as follows:

$$\bar{w}^{(t)} = \tau \bar{w}^{(t-1)} + (1 - \tau) w^{(t)} \tag{8}$$

Model (RErr against AutoAttack [20])	RErr ↓	λ_{max}	$\frac{\lvert\lambda_{min}\rvert}{\lvert\lambda_{max}\rvert}$
AT (baseline)	62.8	1990	0.088
Scaled $\times 0.5$	62.8	7936	0.088
Scaled $\times 2$	62.8	505	0.088
Batch size 8	58.2	3132	0.027
Adam	57.5	540	0.047
Label smoothing	61.2	2484	0.085
Self-supervision	57.1	389	0.041
Entropy-SGD	58.6	5773	0.054
TRADES	56.7	947	0.089
MART	61	1285	0.087
AT-AWP	54.3	1200	0.241

Table A: **Hessian Eigenvalue λ_{max} and Convexity:** For the models from Fig. A and B, we report RErr against AutoAttack [19], the maximum Hessian eigenvalue λ_{max} and the convexity measure of [54] computed as $|\lambda_{min}|/|\lambda_{max}|$. This fraction is supposed to quantify the degree of non-convexity around the found minimum. As can be seen, neither λ_{max} nor convexity correlate well with adversarial robustness. Regarding λ_{max} this is due to the Hessian eigenspectrum not being scale-invariant, as shown for scaled versions ($\times 0.5$ and $\times 2$) of our AT baseline.

where $w^{(t)}$ are the weights in iteration t *after* the gradient update. Weight averaging is motivated by finding the weights $\bar{w}$ in the center of the found local minimum. As, depending on the learning rate, training tends to oscillate, the average of the iterates is assumed to be close to the actual center of the minimum. In our experiments, we consider $\tau \in \{0.98, 0.985, 0.99, 0.9975\}$.

Weight Clipping: Following [85], we implement weight clipping by clipping the weights to $[-w_{max}, w_{max}]$ after each training iteration. We found that w_{max} can be chosen as small as 0.005, which we found to work particularly well. Larger w_{max} does *not* have significant impact on adversarial robustness for AT. [85] argues that weight clipping together with minimizing cross-entropy loss leads to more redundant weights, improving robustness to random weight perturbations. As result, we also expect weight clipping to improve flatness. We consider $w_{max} \in \{0.005, 0.01, 0.025\}$.

Ignoring Incorrect Examples & Preventing Label Leaking: As robust overfitting in AT leads to large RLoss on incorrectly classified test examples, we investigate whether (a) *not* computing adversarial examples on incorrectly classified examples (during training) or (b) computing adversarial examples against the predicted (not true) label (during training) helps to mitigate robust overfitting. These changes can be interpreted as ablations of MART [92] and are easily implemented. Note that option (b) is essentially computing adversarial examples without label leaking [51]. However, as shown in Fig. G, these two variants of AT have little to no impact on robust overfitting.

AutoAugment: In [21], an automatic procedure for finding data augmentation policies is proposed, so-called AutoAugment. We use the found CIFAR10 policy (cf. [21], appendix), which includes quite extreme augmentations. For example, large translations are possible, rendering the image nearly completely uniform, only leaving few pixels at the border. In practice, AutoAugment usually prevents convergence and, thus, avoids overfitting. We further combine AutoAugment with CutOut [23] (using random 16×16 "cutouts"). We apply both AutoAugment and CutOut on top of our standard data augmentation, i.e., random flipping and cropping. We use publicly available PyTorch implementations[1].

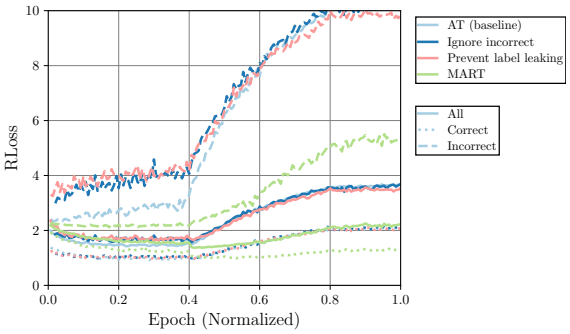


Figure G: **Approaches of Handling Incorrect Examples:** We plot test RLoss on all (solid), correctly classified (dotted) and incorrectly classified (dashed) examples throughout training. We consider our AT baseline (light blue), ignoring incorrectly classified training examples in the RLoss computation during training (dark blue) and preventing label leaking by computing adversarial examples against the *predicted* labels during training (rose). However, these "simple" approaches of tackling the high RLoss on incorrectly classified test examples are not successful in reducing robust overfitting. As outlined in the main paper, MART [92] (green) is able to dampen overfitting through an additional robust KL-loss weighted by confidence, see text.

Entropy-SGD [12] explicitly encourages flatter minima by taking the so-called "local" entropy into account. As a result, Entropy-SGD not only finds "deep" minima (i.e., low loss values) but also flat ones. In practice, this is done using nested SGD: the inner loop approximates the local entropy using stochastic gradient Langevin dynamics (SGLD), the outer loop updates the weights. The number of inner iterations is denoted by L. While the original work [12] uses L in $[5, 20]$ on CIFAR10, we experiment with $L \in \{1, 2, 3, 5\}$. Note that, for fair comparison, we train for $^{150}/_{L}$ epochs. For details on the Entropy-SGD algorithm, we refer to [12]. Our implementation follows the official PyTorch implementation[2].

Activation functions: We consider three recently proposed activation functions: SiLU [28], MiSH [62] and

[1] https://github.com/DeepVoltaire/AutoAugment, https://github.com/uoguelph-mlrg/Cutout

[2] https://github.com/ucla-vision/entropy-sgd

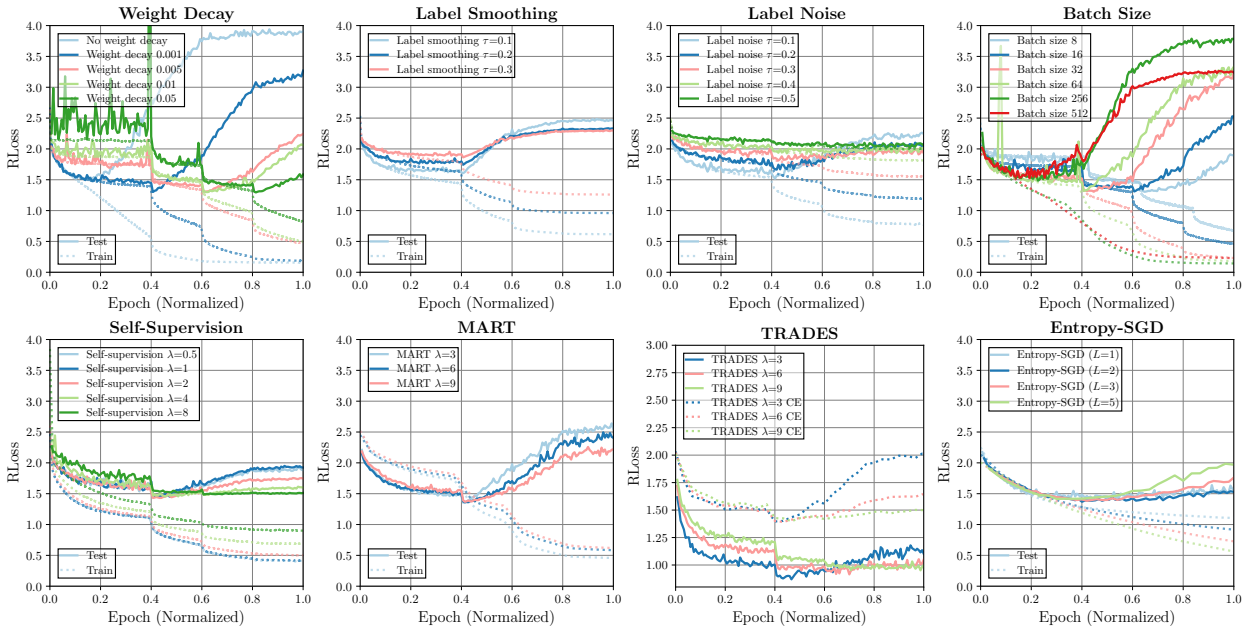


Figure H: **Training Curves for Varying Hyper-Parameters:** We plot RLoss for selected methods and hyper-parameters to demonstrate the impact of hyper-parameters on avoiding or reducing robust overfitting. Note that, for TRADES, we show both RLoss on adversarial examples computed by maximizing the KL-divergence in Eq. (13) (solid) and on adversarial examples obtained by maximizing cross-entropy loss ("CE", dotted).

GeLU [39]. These are defined as:

$$\text{(SiLU)} \quad x\sigma(x) \text{ with } \sigma(x) = 1/(1+\exp(-x)), \tag{9}$$

$$\text{(MiSH)} \quad x\tanh(\log(1+\exp(x))), \tag{10}$$

$$\text{(GeLU)} \quad x\sigma(1.702x). \tag{11}$$

All of these activation functions can be seen as smooth versions of the ReLU activation. In [82], some of these activation functions are argued to avoid robust overfitting due to lower curvature compared to ReLU.

AT-AWP: AT with adversarial weight perturbations (AT-AWP) [96] computes adversarial weight perturbations *on top* of adversarial examples to further regularize training. This is similar to our worst-case flatness measure of RLoss, however, adversarial examples and adversarial weights are computed sequentially, not jointly, and only one iteration is used to compute adversarial weights. Specifically, after computing adversarial examples $\tilde{x} = x + \delta$, an adversarial weight perturbation ν is computed by solving

$$\max_{\nu \in B_\xi(w)} \mathcal{L}(f(\tilde{x}; w+\nu), y) \tag{12}$$

with $B_\xi(w)$ as in Eq. (4) using one iteration of gradient ascent with fixed step size of ξ. The gradient is normalized per layer as in Eq. (2). We considered $\xi \in \{0.0005, 0.001, 0.005, 0.01, 0.015, 0.02\}$ and between 1 and 7 iterations and found that $\xi = 0.01$ and 1 iteration works best (similar to [96]).

TRADES: [102] proposes an alternative formulation of AT that allows a better trade-off between adversarial robustness and (clean) accuracy. The loss to be minimized is

$$\mathcal{L}(f(x;w), y) + \lambda \max_{\|\delta\|_\infty \leq \epsilon} \text{KL}(f(x;w), f(x+\delta;w)). \tag{13}$$

During training, adversarial examples are computed by maximizing the KL-divergence (instead of cross-entropy loss), i.e., using the second term in Eq. (13). Commonly $\lambda = 6$ is chosen, however, we additionally tried $\lambda \in \{1, 3, 6, 9\}$. We follow the official implementation[3].

MART [92] explicitly addresses the problem of incorrectly classified examples during training. First, the cross-entropy loss $\mathcal{L}$ for training is replaced using a binary cross-entropy loss $\mathcal{L}_{\text{bin}}$, i.e., classifying correct class vs. most-confident "other" class:

$$\mathcal{L}_{\text{bin}}(f(x;w), y) = -\log(f_y(x;w)) - \log(1 - \max_{y' \neq y} f_{y'}(x;w)). \tag{14}$$

Second, the KL-divergence used in TRADES in Eq. (13) is combined with a confidence-based weight:

$$\mathcal{L}_{\text{bin}}(f(\tilde{x};w), y) + \lambda \text{KL}(f(x;w), f(\tilde{x};w))(1 - f_y(x;w)) \tag{15}$$

[3] https://github.com/yaodongyu/TRADES

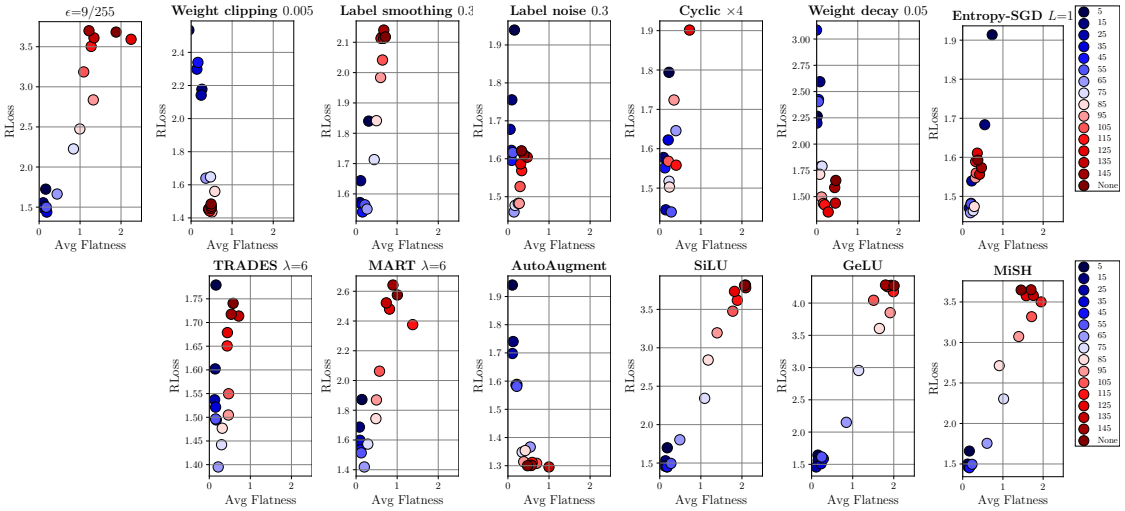


Figure I: **Flatness Throughout Training:** Complementary to the main paper, we plot RLoss against average-case flatness in RLoss for selected methods throughout training epochs. Early epochs are shown in dark blue, late epochs are shown in dark red. For cyclic learning rate, we show 4 cycles with a total of 120 epochs. For many methods not avoiding robust overfitting, flatness decreases alongside an increase in RLoss during overfitting. Using, e.g., AutoAugment, label noise or Entropy-SGD, in contrast, both effects are reduced.

Adversarial examples are still computed by maximizing regular cross-entropy loss. We follow the official implementation[4]. MART is successful in reducing robust overfitting on incorrectly classified examples, as shown in Fig. G.

PGD-τ: In [103], a variant of PGD is proposed for AT: PGD-τ stops maximization τ iterations *after* the label flipped. This is supposed to find "friendlier" adversarial examples that can be used for AT. Note that $\tau = 0$ also does *not* compute adversarial examples on incorrectly classified training examples. We consider $\tau \in \{0, 1, 2, 3\}$.

Self-Supervision: Following [40], we implement AT using rotation-prediction as *additional* self-supervised task. Note, however, that no additional (unlabeled) training examples are used. Specifically, the following learning problem is tackled:

$$\begin{aligned} &\max_{\|\delta\|_\infty \leq \epsilon} \mathcal{L}(f(x+\delta; w), y) \\ &\quad + \lambda \max_{\|\delta\|_\infty \leq \epsilon} \mathcal{L}(f(\text{rot}(x+\delta, r); w), y_r) \\ &\quad r \in \{0, 90, 180, 270\}, y_r \in \{0, 1, 2, 3\} \end{aligned} \tag{16}$$

where $\text{rot}(x, r)$ rotates the training example x by r degrees. In practice, we split every batch in half: The first half uses the original training examples with correct labels. Examples in the second half are rotated randomly by $\{0, 90, 180, 270\}$ degrees, and the labels correspond to the rotation (i.e., $\{0, 1, 2, 3\}$). Adversarial examples are computed against the true or rotation-based labels. Note that, in contrast to common practice [83], we do *not* predict all four possible rotations every batch, but just one randomly drawn per example. We still use 150 epochs in total. We consider $\lambda \in \{0.5, 1, 2, 4, 8\}$.

[4] https://github.com/YisenWang/MART

Additional Unlabeled Examples: As proposed in [11, 1], we also consider additional, pseudo-labeled examples during training. We use the provided pseudo-labeled data from [11] and split each batch in half: using 50% original CIFAR10 training examples, and 50% pseudo-labeled training examples from [11]. We still use 150 epochs in total. We follow the official PyTorch implementation[5].

G.1. Training Curves

Fig. H shows (test) RLoss throughout training for selected methods and hyper-parameters. Across all methods, we found that hyper-parameters have a large impact on robust overfitting. For example, weight decay or smaller batch sizes can reduce and delay robust overfitting considerably if regularization is "strong" enough, i.e., large weight decay or low batch size (to induce more randomness). For the other methods, difference between hyper-parameters is more subtle. However, across all cases, reduced overfitting generally goes hand in hand with higher RLoss on training examples, i.e., the robust generalization gap is reduced. This indicates that avoiding convergence on training examples plays an important role in avoiding robust overfitting.

Training curves for all methods are shown in Fig. K.

G.2. Flatness for Methods

Flatness Throughout Training: Fig. I shows RLoss (y-axis) plotted against average-case flatness in RLoss (x-axis) throughout training, i.e., over epochs (dark blue to dark red), for methods not shown in the main paper. Strikingly, using higher $\epsilon = 9/255$ or alternative activation func-

[5] https://github.com/yaircarmon/semisup-adv

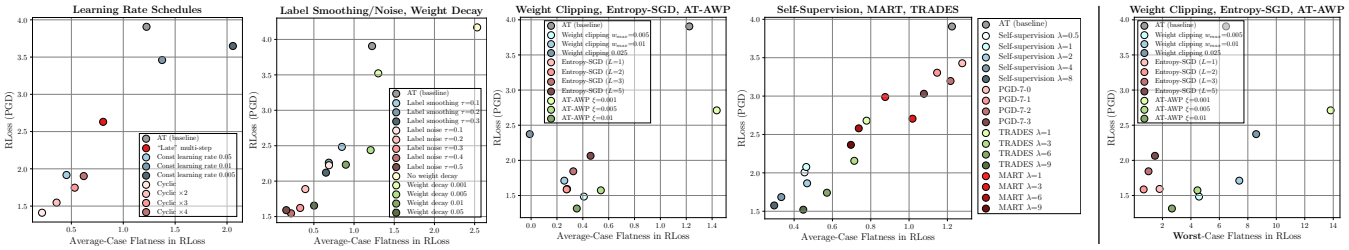


Figure J: **Robustness and Flatness for Varying Hyper-Parameters: Left:** RLoss (y-axis) plotted against average-case flatness of RLoss (x-axis) for various groups of methods: learning rate schedules (left), label smoothing/noise and weight decay (middle left), weight clipping, Entropy-SGD and AT-AWP (middle right) as well as AT with self-supervision, MART and TRADES (right). As outlined in Sec. G, we considered multiple hyper-parameter settings per method and show that favorable hyper-parameters in terms of adversarial robustness also result in improved flatness. That is, in most cases, varying hyper-parameters creates (roughly) a diagonal line in these plots. Interestingly, weight clipping can be considered an outlier: adversarial robustness improves while *average-case* flatness *reduces*. **Right:** RLoss (y-axis) plotted against *worst-case* flatness in RLoss (x-axis). Here, flatness for weight clipping aligns well with RLoss.

tions (SiLU [28], GeLU [39] or MiSH [62]) affect neither robust overfitting nor flatness significantly. Interestingly, as discussed in the main paper, label smoothing avoids sharper minima during overfitting, but does *not* avoid an increased RLoss. Methods that consistently reduce or avoid robust overfitting, e.g., weight clipping, label noise, strong weight decay or AutoAugment, avoid both the increase in RLoss as well as worse flatness. Clearly, the observations from the main paper are confirmed: flatness usually reduces alongside RLoss in robust overfitting.

Flatness Across Hyper-Parameters: In Fig. J, we consider flatness when changing hyper-parameters of selected methods. As before, we plot RLoss (y-axis) against average-case flatness in RLoss (x-axis) for various groups of methods: learning rate schedules (first column), label smoothing/noise and weight decay (second column), methods explicitly improving flatness, i.e., weight clipping, Entropy-SGD and AT-AWP (third column), as well as self-supervision, MART and TRADES (fourth column). Except for weight clipping, hyper-parameter settings with improved adversarial robustness also favor flatter minima. In most cases, this relationship follows a clear, diagonal line. For weight clipping, in contrast, the relationship is reversed: improved flatness reduces RLoss. Thus, Fig. J (fifth column) considers worst-case flatness in RLoss. Here, "stronger" weight clipping improves both robustness *and* flatness. This supports our discussion in the main paper: methods need at least "some kind" of flatness, average- or worst-case, in order to improve adversarial robustness.

H. Results in Tabular Form

Tab. D and E report the quantitative results from all our experiments. Besides flatness in RLoss, we also report both average- and worst-case flatness in (clean) Loss. As described in the main paper, we use $\xi = 0.5$ for average-case flatness and $\xi = 0.003$ for worst-case flatness. In Tab. D, methods are sorted (in ascending order) by RErr against AutoAttack [20]. Additionally, we split all methods into four groups: good, average, poor and worse robustness at 57%, 60% and 62.8% RErr. These thresholds correspond roughly to the 30% and 70% percentile of all methods with RErr $\leq 62.8\%$. As our AT baseline obtains 62.8% RErr, we group all methods with higher RErr than 62.8% in worse robustness. In Tab. E, methods are sorted (in ascending order) by RLoss against PGD. Finally, Tab. B and C report RErr and RLoss, together with our average- and worst-case flatness (of RLoss) measures for the evaluated, pre-trained models from RobustBench [18].

Model	Test Robustness			Train Robustness			Flatness	
(sorted by RLoss on AA) (PGD = PGD-20, 10 restarts) (AA = AutoAttack [19])	Err (test)	RErr (test) (PGD)	RErr (test) (AA)	Err (train)	RErr (train) (PGD)	RErr (train) (AA)	Avg RLoss	Worst RLoss
Carmon et al. [7]	10.31	37.6	40.8	1.93	16.8	19.2	0.7	0.34
Engstrom et al. [16]	12.97	45.3	49.2	6.71	33.1	36.3	0.23	0.51
Pang et al. [43]	14.87	36.6	45.8	7.79	20.5	28.6	0.08	0.07
Wang [60]	12.5	37.1	42.8	8.07	24.8	32.1	0.61	0.34
Wong et al. [61]	16.66	54.4	57.6	11.86	44.9	49.2	0.3	0.16
Wu et al. [62]	14.64	41.5	43.9	2.2	14.5	16.5	0.49	0.09
Zhang et al. [72]	15.08	44.1	46.4	7.83	29.9	33.6	0.61	0.43
Zhang et al. [73]	15.48	43	47.2	4.85	26.3	30.1	0.51	0.13

Table B: **RobustBench [18]: Err, RErr and Flatness in RLoss:** Err and RErr on train and test examples as well as average- and worst-case flatness in RLoss for pre-trained models from RobustBench. In contrast to Tab. D, the RobustBench models were obtained using early stopping.

Model	Test Robustness			Train Robustness			Flatness	
(sorted by RLoss on AA) (PGD = PGD-20, 10 restarts) (AA = AutoAttack [19])	Loss (test)	RLoss (test) (PGD)	RLoss (test) (AA)	Loss (train)	RLoss (train) (PGD)	RLoss (train) (AA)	Avg RLoss	Worst RLoss
Carmon et al. [7]	0.53	1.02	0.63	0.36	0.62	0.41	0.7	0.34
Engstrom et al. [16]	0.44	1.25	0.59	0.29	0.82	0.41	0.23	0.51
Pang et al. [43]	1.84	1.98	1.86	1.8	1.91	1.8	0.08	0.07
Wang [60]	0.64	1.11	0.73	0.54	0.9	0.6	0.61	0.34
Wong et al. [61]	0.57	1.37	0.73	0.46	1.11	0.61	0.3	0.16
Wu et al. [62]	0.63	1.13	0.72	0.37	0.61	0.41	0.49	0.09
Zhang et al. [72]	0.55	1.19	0.66	0.39	0.83	0.48	0.61	0.43
Zhang et al. [73]	0.85	1.27	0.93	0.71	1.01	0.76	0.51	0.13

Table C: **RobustBench [18]: *Loss, RLoss* and Flatness in RLoss:** Loss and RLoss on train and test examples as well as average- and worst-case flatness in RLoss for pre-trained models from RobustBench. In contrast to Tab. E, the RobustBench models were obtained using early stopping.

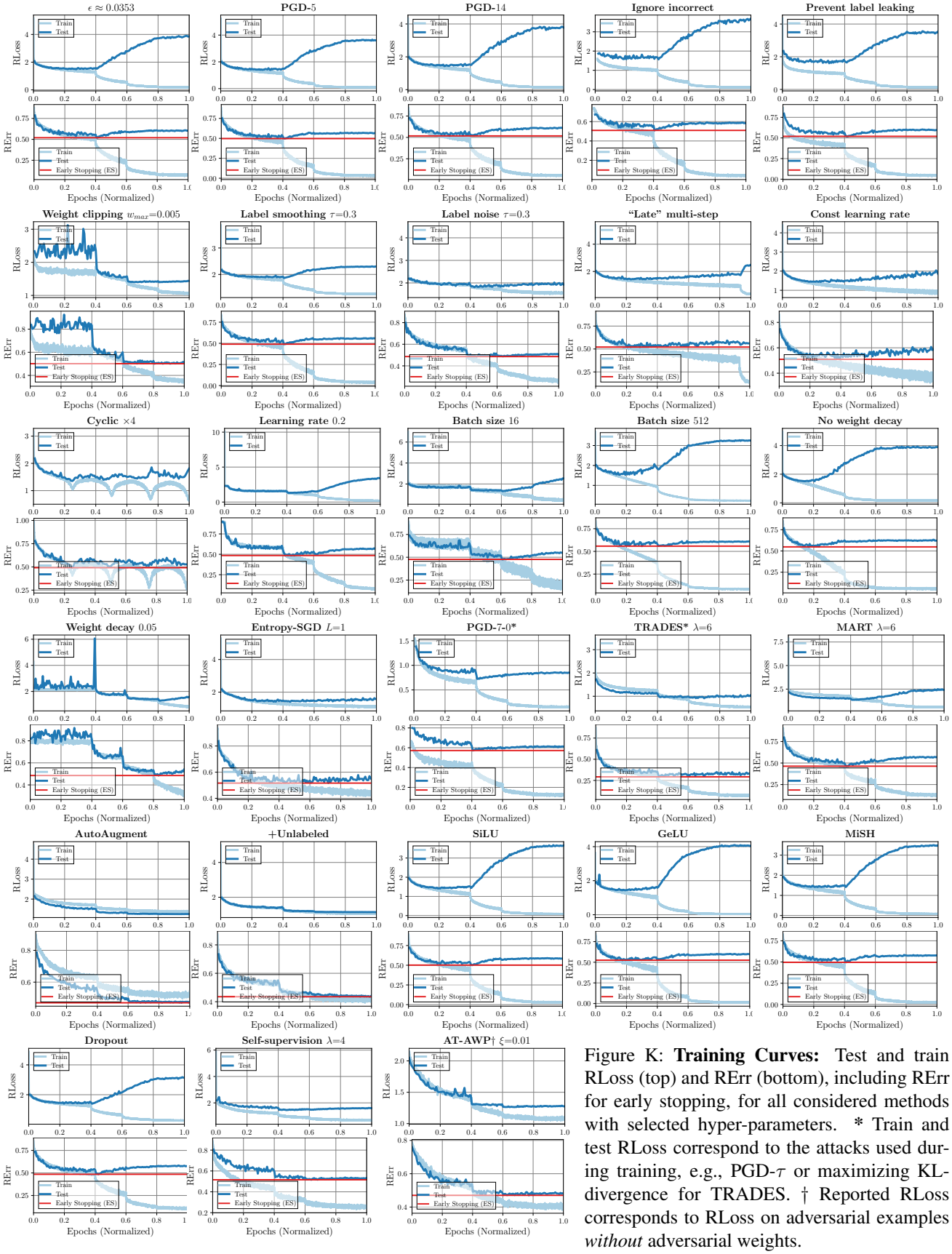


Figure K: **Training Curves:** Test and train RLoss (top) and RErr (bottom), including RErr for early stopping, for all considered methods with selected hyper-parameters. * Train and test RLoss correspond to the attacks used during training, e.g., PGD-τ or maximizing KL-divergence for TRADES. † Reported RLoss corresponds to RLoss on adversarial examples *without* adversarial weights.

Model	Test Robustness			Train Robustness			Early Stopping		Flatness			
(sorted by RErr on AA) (PGD = PGD-20, 10 restarts) (AA = AutoAttack [19])	Err (test)	RErr (test) (PGD)	RErr (test) (AA)	Err (train)	RErr (train) (PGD)	RErr (train) (AA)	RErr (stop) (PGD)	RErr (stop) (AA)	Avg Loss	Worst Loss	Avg **RLoss**	Worst **RLoss**
+Unlabeled	16.96	45.9	48.9	12.6	38.6	43.2	45.3	48.9	0.12	4.64	0.32	1.2
Cyclic $\times 2$	19.66	51.2	53.6	7.64	32.3	35.4	51	53.6	0.09	3.93	0.35	1.5
AutoAugment	16.89	49.5	54.0	12.25	42.8	47.9	49.5	53.5	0.13	15.01	0.49	0.69
AT-AWP ξ=0.01	21.4	50.7	54.3	13.52	37.4	43.1	48.9	53.6	0.12	6.17	0.35	2.68
AT-AWP ξ=0.005	20.05	52.5	55	7.34	28.1	31.8	50.8	53.3	0.15	6.98	0.54	4.46
Label noise τ=0.4	20.56	52.4	55	9.66	32.8	36.8	51.2	54.8	0.11	3.95	0.21	0.96
TRADES λ=9	23.03	52.4	55	2.92	16.4	18.8	49.7	53	0.19	5.04	0.45	3.08
Cyclic $\times 3$	20.04	53.1	55.2	5.62	26.9	30.6	53.1	55.2	0.1	4.1	0.53	0.93
Cyclic	22.42	53.2	55.4	13.09	39.5	43.5	53.2	55.4	0.07	2.6	0.22	0.41
Label noise τ=0.5	22.71	51.3	55.4	15.04	40.4	45.5	51.3	55.4	0.09	0.43	0.16	0.13
Label noise τ=0.3	19.9	54.2	56.2	5.47	26.9	30	51.8	55.5	0.15	3.37	0.33	0.93
Weight clipping w_{max}=0.005	21.39	54.1	56.5	10.19	35.6	39	54.1	56.5	0.74	10.49	0.41	4.58
TRADES λ=6	21.68	55.3	56.7	1.74	13.5	15.8	50.1	53.4	0.21	5.12	0.57	2.26
Cyclic $\times 4$	19.85	55.2	56.9	4.01	23.1	26	55.1	56.9	0.16	6.65	0.62	0.8
Self-supervision λ=4	17.1	55.3	57.1	5.76	41.9	45	55.3	56.8	0.12	5.59	0.34	2.64
Adam	25.84	53.9	57.5	18.87	47.9	52.3	53.9	57.5	0.22	2.65	0.56	0.9
Entropy-SGD (L=2)	24.53	54.4	57.6	9.03	35.4	38.8	52.6	55.2	0.08	1.76	0.27	0.7
Self-supervision λ=1	15.9	56.9	58.1	1.48	28.3	31.6	55.9	57.5	0.12	6.98	0.46	3.87
Weight decay 0.05	19.32	56.2	58.1	5.03	29	32.8	52	54.8	0.12	5.77	0.51	3.94
Batch size 8	17.73	57.1	58.2	3.46	26.8	31.4	55.6	58.2	0.32	24.01	1.55	12.27
Entropy-SGD (L=1)	25.42	56	58.6	12.79	42.8	46.1	53.2	56.9	0.09	3.24	0.28	1.8
Self-supervision λ=0.5	16.16	58	58.6	1.26	28	30.7	56.7	58.3	0.1	6.48	0.45	3.29
AT-AWP ξ=0.001	18.75	57.3	58.7	1.34	15.1	18.3	52.1	54.6	0.34	20.42	1.44	13.82
Self-supervision λ=2	15.72	57.4	58.7	2.47	33.4	36.6	55.8	57.7	0.1	21.79	0.47	3.47
MART λ=9	22.06	57	58.8	3.86	16	22	50	55	0.18	8.08	0.7	3.42
Weight decay 0.01	18.52	57.2	58.9	2.06	20.1	23.2	51.7	55.3	0.25	16.46	0.9	7.19
Batch size 16	18.12	58.3	59	1.82	20.4	24.5	52.5	55.6	0.33	22.11	1.41	11.39
Self-supervision λ=8	19.6	56.6	59	12.08	50	53.3	56.6	58.6	0.11	3.59	0.29	1.76
TRADES λ=3	20.51	57.7	59.1	0.94	13.4	15.5	52.3	54.9	0.2	19.08	0.71	3.48
Weight decay 0.005	18.79	58.2	59.4	2.03	20.2	23.9	51.8	54.3	0.26	19.67	1.2	8.35
Label noise τ=0.2	19.45	57.5	59.5	2.34	18.8	22.2	50.2	53	0.18	9.79	0.39	1.4
MART λ=3	20.89	58.9	59.6	1.94	14.4	19.2	53.3	57.4	0.17	10.53	1.01	3.99
Weight clipping w_{max}=0.01	19.15	58	59.6	3.28	21.5	24.8	56.7	58.5	0.66	15.1	0.26	7.41
Learning rate 0.2	19.17	58.3	59.7	0.46	9.4	12.4	54.3	56.6	0.2	24.41	1.44	5.75
MiSH	19.29	58.9	59.8	0.06	4.5	5.3	51.8	53.7	0.25	10.04	1.58	3.55
"Late" multi-step	20.63	58.5	59.8	1.6	16.4	18.4	54.2	57.8	0.17	5.24	0.81	2.96
SiLU	19.45	59.7	60	0.07	4.8	5.6	51.3	53.7	0.3	9.97	1.68	4.2
Weight averaging (τ=0.9975)	19.63	59.7	60	0.19	7.9	10	50.5	53	0.23	15.66	1.29	6
Weight clipping 0.025	18.91	59.2	60.4	0.73	12.5	15.6	52.1	54.9	0.32	17.4	0	8.61
Batch size 32	18.72	59.6	60.5	0.56	12	14.6	53.7	55.6	0.18	19.34	1.22	7.88
Entropy-SGD (L=3)	24	58.5	60.5	5.25	29.9	33.9	56.7	59.3	0.09	2.91	0.33	1.03
Label noise τ=0.1	19.39	59	60.8	1.12	14.1	17.5	51.9	55	0.2	16.75	0.69	3.55
Larger ϵ=9/255	21.3	60.4	60.9	0.47	8.9	11.1	51.3	53.8	0.21	10.26	1.34	5.85
Label smoothing τ=0.1	19.55	59.6	61	0.2	6.4	8.5	52.5	55	0.26	8.87	0.85	2.66
MART λ=6	21.51	58.7	61	3.21	16.1	20.8	49.2	54.7	0.18	13.52	0.74	3.17
Weight averaging (τ=0.98)	20.01	60.6	61	0.2	7.6	9.9	54.3	56.3	0.23	12.8	1.37	5.6
Weight decay 0.001	19.47	59.9	61	0.36	10.4	13.3	52	54.8	0.24	8.36	1.3	6.78
Batch size 64	19.06	60.5	61.1	0.3	9.2	11.1	51.2	54.4	0.18	23.13	1.14	5.96
GeLU	20.64	60.8	61.1	0.01	2.7	3.2	54.9	56.7	0.23	14.31	1.56	4.13
Label smoothing τ=0.3	19.41	59.4	61.2	0.27	5.7	8	51.1	54	0.29	18.42	0.65	2.72
MART λ=1	20.51	59.4	61.2	1.04	11.4	14.7	50.3	55.4	0.17	7.97	0.87	3.1
Weight averaging (τ=0.99)	20.41	60.3	61.4	0.19	7.8	9.6	51.7	54.2	0.22	6.12	1.44	4.98
Dropout	18.91	60.5	61.6	0.58	13	16.7	51.2	54.5	0.2	13.81	1.52	7.01
PGD-14	20.8	60.6	61.6	0.22	7.1	9.3	53.6	56.1	0.27	20.9	1.48	5.35
Entropy-SGD (L=5)	23.48	59.5	61.7	3.01	22.2	25.9	53.2	56.6	0.1	3.57	0.46	1.49
Ignore incorrect	18.4	60.5	61.8	0.06	6.3	9	54.4	56.4	0.21	14.65	1.68	5.93
Learning rate 0.1	19.23	61.1	61.9	0.26	8.9	11.5	51.9	54.2	0.21	17.63	1.23	5.26
TRADES λ=1	17.54	59.5	61.9	0.15	16.6	20.7	56.6	59.6	0.16	12.68	0.78	4.3
Weight averaging (τ=0.985)	20.27	61.7	62.3	0.18	7.4	9.4	55.9	58	0.22	15.66	1.35	6.51
Label smoothing τ=0.2	20.07	60.2	62.4	0.26	5.1	7.8	51.9	54.6	0.28	9.94	0.69	2.61
Prevent label leaking	18.38	62.1	62.4	0.38	8.6	10.8	55.3	57.7	0.22	14.62	1.48	6
AT (baseline)	20.2	61	62.8	0.33	8.5	10.7	52.3	54.6	0.21	21.05	1.22	6.49
Const learning rate 0.05	24.96	60.7	62.9	6.17	32.9	37.8	55.4	58.9	0.09	3.52	0.44	0.9
PGD-5	20.22	61.8	62.9	0.11	7.3	10.4	55.1	57.4	0.17	20.4	1.24	4.19
Batch size 256	20.86	62.6	63.3	0.28	8.2	10.3	56.9	58.4	0.3	11.22	1.35	8.33
PGD-7-3	17.17	61.7	63.3	0.08	19.5	25.2	51.3	58.8	0.17	7.4	1.08	5.29
Batch size 512	22.58	62.9	63.5	0.64	11	14.2	58.6	60	0.48	23.97	1.92	16.22
Learning rate 0.01	22.83	63	63.5	1.05	15.2	18	57.8	59.7	0.56	23.42	2.25	16.02
No weight decay	23.37	64.8	65.7	0.23	9.2	12.7	57.1	60.3	0.66	21.05	2.53	11.75
PGD-7-0	14.67	63.8	65.7	0.09	23.7	30	59.4	61.4	0.11	6.86	1.28	2.8
PGD-7-2	16.19	63.6	65.9	0.1	20.9	28.1	58.3	62.3	0.14	22.81	1.21	2.55
PGD-7-1	15.02	64.1	67.1	0.11	25.9	34.3	58.8	63.8	0.13	11.71	1.15	2.33
Const learning rate 0.01	25.87	66.7	67.4	0.67	18.5	20.7	58.4	61	0.33	15.09	1.37	8.27
Const learning rate 0.005	27.24	68.3	69.2	0.42	15.5	16.7	61.1	65.5	0.59	20.63	2.06	15.74

Table D: **Results: Err, RErr and Flatness in Loss and RLoss.** Err and RErr (PGD-20 and AutoAttack [20]) on test and train examples, together with average- and worst-case flatness in (clean) Loss and RLoss. Methods sorted by (test) RErr against AutoAttack and split into good, average, poor and worse robustness at 57%, 60% and 62.8% RErr, see text.

Model	Test Robustness			Train Robustness			Early Stopping	
(sorted by RLoss on PGD) (PGD = PGD-20, 10 restarts) (AA = AutoAttack [19])	Loss (test)	RLoss (test) (PGD)	RLoss (test) (AA)	Loss (train)	RLoss (train) (PGD)	RLoss (train) (AA)	RLoss (stop) (PGD)	RLoss (stop) (AA)
+Unlabeled	0.57	1.18	0.67	0.47	0.94	0.56	1.18	0.67
AutoAugment	0.58	1.3	0.71	0.48	1.08	0.61	1.3	0.71
AT-AWP ξ=0.01	0.7	1.31	0.81	0.55	0.99	0.62	1.3	0.81
Cyclic	0.68	1.41	0.8	0.49	0.97	0.58	1.41	0.8
Adam	0.8	1.46	0.89	0.66	1.19	0.74	1.45	0.89
Weight clipping w_{max}=0.005	0.77	1.48	0.91	0.53	0.99	0.62	1.47	0.9
TRADES λ=9	0.77	1.52	0.9	0.33	0.58	0.37	1.42	0.9
Label noise τ=0.4	0.93	1.55	1.05	0.71	1.15	0.8	1.5	1.05
Cyclic $\times$2	0.6	1.55	0.74	0.32	0.76	0.42	1.55	0.74
AT-AWP ξ=0.005	0.59	1.57	0.74	0.29	0.66	0.38	1.36	0.74
Self-supervision λ=8	0.59	1.58	0.76	0.43	1.24	0.62	1.57	0.76
Entropy-SGD (L=2)	0.72	1.59	0.83	0.4	0.87	0.5	1.44	0.83
Label noise τ=0.5	1.12	1.59	1.22	1	1.39	1.08	1.59	1.22
Entropy-SGD (L=1)	0.77	1.59	0.87	0.5	1.06	0.6	1.44	0.87
Label noise τ=0.3	0.78	1.62	0.94	0.45	0.91	0.55	1.47	0.94
Weight decay 0.05	0.61	1.65	0.78	0.28	0.73	0.39	1.33	0.78
Self-supervision λ=4	0.51	1.68	0.71	0.25	1.02	0.45	1.62	0.71
Weight clipping w_{max}=0.01	0.62	1.71	0.83	0.22	0.61	0.31	1.59	0.83
TRADES λ=6	0.7	1.74	0.86	0.2	0.44	0.25	1.4	0.86
Cyclic $\times$3	0.6	1.75	0.74	0.24	0.65	0.34	1.59	0.74
Entropy-SGD (L=3)	0.69	1.84	0.85	0.26	0.72	0.38	1.57	0.85
Self-supervision λ=2	0.47	1.86	0.69	0.13	0.8	0.33	1.53	0.69
Label noise τ=0.2	0.68	1.89	0.9	0.22	0.63	0.32	1.4	0.9
Cyclic $\times$4	0.6	1.9	0.78	0.2	0.57	0.28	1.44	0.78
Const learning rate 0.05	0.75	1.92	0.89	0.31	0.81	0.43	1.54	0.88
Self-supervision λ=0.5	0.48	2.01	0.71	0.09	0.67	0.27	1.6	0.71
Entropy-SGD (L=5)	0.71	2.06	0.89	0.18	0.57	0.28	1.5	0.86
Self-supervision λ=1	0.48	2.08	0.72	0.09	0.67	0.27	1.63	0.7
Label smoothing τ=0.3	0.77	2.12	1.02	0.15	0.47	0.21	1.46	0.97
TRADES λ=3	0.65	2.16	0.85	0.1	0.34	0.17	1.42	0.83
Batch size 8	0.56	2.22	0.78	0.17	0.64	0.3	1.86	0.76
Label noise τ=0.1	0.63	2.22	0.87	0.1	0.42	0.19	1.37	0.86
Weight decay 0.01	0.58	2.23	0.78	0.12	0.47	0.22	1.35	0.78
Label smoothing τ=0.2	0.71	2.26	0.98	0.09	0.35	0.14	1.44	0.89
MART λ=9	0.7	2.36	0.93	0.24	0.48	0.3	1.41	0.93
Weight clipping 0.025	0.57	2.37	0.81	0.06	0.32	0.14	1.39	0.81
Weight decay 0.005	0.58	2.44	0.8	0.11	0.46	0.23	1.43	0.76
Label smoothing τ=0.1	0.64	2.48	0.88	0.04	0.24	0.09	1.43	0.8
MART λ=6	0.71	2.58	0.93	0.21	0.45	0.27	1.4	0.93
"Late" multi-step	0.66	2.63	0.87	0.09	0.36	0.17	1.47	0.87
TRADES λ=1	0.56	2.68	0.81	0.04	0.38	0.18	1.74	0.74
MART λ=3	0.69	2.71	0.89	0.14	0.38	0.21	1.48	0.89
AT-AWP ξ=0.001	0.62	2.71	0.84	0.08	0.34	0.16	1.35	0.78
Batch size 16	0.57	2.78	0.83	0.1	0.46	0.22	1.63	0.74
Learning rate 0.01	0.72	2.94	0.96	0.07	0.34	0.16	1.74	0.85
MART λ=1	0.7	2.99	0.94	0.08	0.29	0.15	1.44	0.94
PGD-7-3	0.55	3.03	0.8	0.04	0.48	0.19	1.7	0.73
PGD-7-2	0.54	3.2	0.79	0.03	0.48	0.21	2.12	0.71
PGD-7-1	0.51	3.3	0.75	0.04	0.61	0.25	2.43	0.68
Batch size 512	0.77	3.41	1.04	0.05	0.27	0.12	1.86	0.85
PGD-7-0	0.49	3.43	0.74	0.03	0.58	0.21	2.48	0.65
Dropout	0.66	3.44	0.9	0.04	0.3	0.13	1.44	0.76
Const learning rate 0.01	0.89	3.46	1.07	0.04	0.43	0.17	1.67	0.97
Weight decay 0.001	0.69	3.52	0.92	0.03	0.24	0.1	1.41	0.77
Batch size 32	0.67	3.54	0.91	0.04	0.27	0.11	1.69	0.73
Batch size 64	0.69	3.62	0.93	0.03	0.22	0.09	1.44	0.76
Const learning rate 0.005	0.97	3.65	1.24	0.04	0.35	0.14	1.62	1.12
MiSH	0.69	3.65	0.92	0.01	0.1	0.04	1.45	0.73
Learning rate 0.1	0.7	3.65	0.94	0.02	0.19	0.09	1.39	0.79
Learning rate 0.2	0.72	3.66	0.97	0.03	0.21	0.1	1.67	0.75
Larger ϵ=9/255	0.78	3.69	1.01	0.03	0.19	0.09	1.45	0.79
Prevent label leaking	0.67	3.76	0.91	0.02	0.21	0.08	1.74	0.7
SiLU	0.71	3.81	0.96	0.01	0.11	0.04	1.47	0.73
PGD-14	0.76	3.84	1.05	0.02	0.15	0.07	1.52	0.78
Weight averaging (τ=0.9975)	0.73	3.88	1	0.02	0.17	0.08	1.34	0.81
AT (baseline)	0.75	3.91	0.99	0.02	0.19	0.08	1.53	0.77
Weight averaging (τ=0.98)	0.75	3.94	1.05	0.02	0.16	0.08	1.96	0.8
Weight averaging (τ=0.985)	0.75	3.95	1	0.02	0.16	0.07	1.94	0.77
Ignore incorrect	0.71	3.99	0.96	0.01	0.16	0.07	1.65	0.7
Weight averaging (τ=0.99)	0.76	3.99	1.07	0.02	0.18	0.07	1.53	0.74
Batch size 256	0.79	4.02	1.07	0.02	0.18	0.08	1.74	0.81
No weight decay	0.9	4.17	1.15	0.02	0.22	0.1	1.55	0.91
PGD-5	0.77	4.22	0.99	0.01	0.17	0.08	1.62	0.77
GeLU	0.82	4.27	1.06	0	0.06	0.02	1.7	0.79

Table E: **Results: *Loss* and *RLoss*.** Loss and RLoss (PGD-20 and AutoAttack [20]) on test and train examples corresponding to the results in Tab. D.

References

[1] Jean-Baptiste Alayrac, Jonathan Uesato, Po-Sen Huang, Alhussein Fawzi, Robert Stanforth, and Pushmeet Kohli. Are labels required for improving adversarial robustness? In *NeurIPS*, 2019. 9

[2] Laurent Amsaleg, James Bailey, Dominique Barbe, Sarah M. Erfani, Michael E. Houle, Vinh Nguyen, and Milos Radovanovic. The vulnerability of learning to adversarial perturbation increases with intrinsic dimensionality. In *WIFS*, 2017. 1

[3] Anish Athalye and Nicholas Carlini. On the robustness of the CVPR 2018 white-box adversarial example defenses. *arXiv.org*, abs/1804.03286, 2018. 1

[4] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv.org*, abs/1802.00420, 2018. 1

[5] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. Dimensionality reduction as a defense against evasion attacks on machine learning classifiers. *arXiv.org*, abs/1704.02654, 2017. 1

[6] Wieland Brendel and Matthias Bethge. Comment on "biologically inspired protection of deep networks from adversarial attacks". *arXiv.org*, abs/1704.01547, 2017. 1

[7] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *ICLR*, 2018. 1

[8] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *AISec*, 2017. 1

[9] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *SP*, 2017. 1

[10] Nicholas Carlini and David A. Wagner. Defensive distillation is not robust to adversarial examples. *arXiv.org*, abs/1607.04311, 2016. 1

[11] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C. Duchi, and Percy Liang. Unlabeled data improves adversarial robustness. In *NeurIPS*, 2019. 1, 9

[12] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer T. Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. In *ICLR*, 2017. 1, 7

[13] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *AISec*, 2017. 1

[14] Nicholas Cheney, Martin Schrimpf, and Gabriel Kreiman. On the robustness of convolutional neural networks to internal architecture and weight perturbations. *arXiv.org*, abs/1703.08245, 2017. 1

[15] Ching-Tai Chiu, Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka. Training techniques to obtain fault-tolerant neural networks. In *Annual International Symposium on Fault-Tolerant Computing*, 1994. 1

[16] Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv.org*, abs/1902.02918, 2019. 1

[17] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. *arXiv.org*, abs/1810.07481, 2018. 1

[18] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. *arXiv.org*, abs/2010.09670, 2020. 10, 11

[19] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. *arXiv.org*, abs/2003.01690, 2020. 5, 6, 7, 11, 13, 14

[20] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020. 7, 10, 13, 14

[21] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *arXiv.org*, abs/1805.09501, 2018. 1, 7

[22] Dipti Deodhare, M. Vidyasagar, and S. Sathiya Keerthi. Synthesis of fault-tolerant feedforward neural networks using minimax optimization. *TNN*, 9(5):891–900, 1998. 1

[23] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv.org*, abs/1708.04552, 2017. 7

[24] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *ICML*, 2017. 5

[25] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *CVPR*, 2018. 1, 6

[26] Vasisht Duddu, D. Vijay Rao, and Valentina E. Balas. Adversarial fault tolerant training for deep neural networks. *arXiv.org*, abs/1907.03103, 2019. 2

[27] Jacob Dumford and Walter J. Scheirer. Backdooring convolutional neural networks via targeted weight perturbations. *arXiv.org*, abs/1812.03128, 2018. 1

[28] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *NN*, 107, 2018. 7, 10

[29] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv.org*, abs/1703.00410, 2017. 1

[30] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. In *SP*, pages 3–18, 2018. 1

[31] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv.org*, abs/1412.6572, 2014. 1

[32] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy A. Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv.org*, abs/1810.12715, 2018. 1

[33] Sven Gowal, Chongli Qin, Jonathan Uesato, Timothy A. Mann, and Pushmeet Kohli. Uncovering the limits of adversarial training against norm-bounded adversarial examples. *arXiv.org*, abs/2010.03593, 2020. 1, 6

[34] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv.org*, abs/1702.06280, 2017. 1

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5, 6

[36] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defense: Ensembles of weak defenses are not strong. In *USENIX Workshops*, 2017. 1

[37] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. Defending and harnessing the bit-flip based adversarial weight attack. In *CVPR*, 2020. 1

[38] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NeurIPS*, 2017. 1

[39] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv.org*, abs/1606.08415, 2016. 8, 10

[40] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty. In *NeurIPS*, 2019. 1, 9

[41] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. *arXiv.org*, abs/1807.07978, 2018. 1

[42] Andrew Ilyas, Ajil Jalal, Eirini Asteri, Constantinos Daskalakis, and Alexandros G. Dimakis. The robust manifold defense: Adversarial training using generative models. *arXiv.org*, abs/1712.09196, 2017. 1

[43] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 3, 5, 6

[44] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *UAI*, 2018. 6

[45] Daniel Jakubovitz and Raja Giryes. Improving DNN robustness to adversarial attacks using jacobian regularization. *arXiv.org*, abs/1803.08680, 2018. 1

[46] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. Model reuse attacks on deep learning systems. In *CCS*, 2018. 1

[47] Harini Kannan, Alexey Kurakin, and Ian J. Goodfellow. Adversarial logit pairing. *arXiv.org*, abs/1803.06373, 2018. 1

[48] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 5

[49] Aounon Kumar, A. Levine, S. Feizi, and T. Goldstein. Certifying confidence via randomized smoothing. *ArXiv*, abs/2009.08061, 2020. 1

[50] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv.org*, abs/1607.02533, 2016. 1

[51] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv.org*, abs/1611.01236, 2016. 7

[52] Alex Lamb, Jonathan Binas, Anirudh Goyal, Dmitriy Serdyuk, Sandeep Subramanian, Ioannis Mitliagkas, and Yoshua Bengio. Fortified networks: Improving the robustness of deep networks by modeling the manifold of hidden representations. *arXiv.org*, abs/1804.02485, 2018. 1

[53] Guang-He Lee, David Alvarez-Melis, and Tommi S. Jaakkola. Towards robust, locally linear deep networks. *arXiv.org*, abs/1907.03207, 2019. 1

[54] Hao Li, Zheng Xu, G. Taylor, and T. Goldstein. Visualizing the loss landscape of neural nets. In *NeurIPS*, 2018. 1, 2, 5, 7

[55] Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Defense against adversarial attacks using high-level representation guided denoiser. In *CVPR*, 2018. 1

[56] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. *arXiv.org*, abs/1712.00673, 2017. 1

[57] Bo Luo, Yannan Liu, Lingxiao Wei, and Qiang Xu. Towards imperceptible and robust adversarial example attacks against neural networks. In *AAAI*, 2018. 1

[58] Xingjun Ma, Bo Li, Yisen Wang adn Sarah M. Erfani, Sudanthi Wijewickrema, Michael E. Houle, Grant Schoenebeck, Dawn Song, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv.org*, abs/1801.02613, 2018. 1

[59] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv.org*, abs/1706.06083, 2017. 1

[60] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv.org*, abs/1702.04267, 2017. 1

[61] Matthew Mirman, Timon Gehr, and Martin T. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *ICML*, pages 3575–3583, 2018. 1

[62] Diganta Misra. Mish: A self regularized non-monotonic activation function. In *BMVC*, 2020. 7, 10

[63] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *CVPR*, 2016. 1

[64] Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. In *CVPR Workshops*, 2017. 1

[65] Aran Nayebi and Surya Ganguli. Biologically inspired protection of deep networks from adversarial attacks. *arXiv.org*, abs/1703.09202, 2017. 1

[66] Chalapathy Neti, Michael H. Schneider, and Eric D. Young. Maximally fault tolerant neural networks. *TNN*, 3(1):14–23, 1992. 1

[67] Tianyu Pang, Xian Yang, Yinpeng Dong, Hang Su, and Jun Zhu. Bag of tricks for adversarial training. *arXiv.org*, abs/2010.00467, 2020. 6

[68] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *SP*, 2016. 1

[69] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NeurIPS Workshops*, 2017. 6

[70] Rama Chellappa Pouya Samangouei, Maya Kabkab. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. *ICLR*, 2018. 1

[71] Aaditya Prakash, Nick Moran, Solomon Garber, Antonella DiLillo, and James A. Storer. Protecting JPEG images against adversarial attacks. In *DCC*, 2018. 1

[72] Chongli Qin, James Martens, Sven Gowal, Dilip Krishnan, Krishnamurthy Dvijotham, Alhussein Fawzi, Soham De, Robert Stanforth, and Pushmeet Kohli. Adversarial robustness through local linearization. In *NeurIPS*, 2019. 6

[73] Faiz Ur Rahman, Bhavan Vasu, and Andreas E. Savakis. Resilience and self-healing of deep convolutional object detectors. In *ICIP*, 2018. 2

[74] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *ICCV*, 2019. 1

[75] Leslie Rice, Eric Wong, and J. Zico Kolter. Overfitting in adversarially robust deep learning. In *ICML*, 2020. 1

[76] Andrew Slavin Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *AAAI*, 2018. 1

[77] Sayantan Sarkar, Ankan Bansal, Upal Mahbub, and Rama Chellappa. UPSET and ANGRI : Breaking high performance image classifiers. *arXiv.org*, abs/1707.01159, 2017. 1

[78] Lukas Schott, Jonas Rauber, Wieland Brendel, and Matthias Bethge. Robust perception through analysis by synthesis. *arXiv.org*, abs/1805.09190, 2018. 1

[79] Shiwei Shen, Guoqing Jin, Ke Gao, and Yongdong Zhang. Ape-gan: Adversarial perturbation elimination with gan. *arXiv.org*, abs/1707.05474, 2017. 1

[80] Carl-Johann Simon-Gabriel, Yann Ollivier, Bernhard Schölkopf, Léon Bottou, and David Lopez-Paz. Adversarial vulnerability of neural networks increases with input dimension. *arXiv.org*, abs/1802.01421, 2018. 1

[81] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. Fast and effective robustness certification. In *NeurIPS*, pages 10825–10836, 2018. 1

[82] Vasu Singla, Sahil Singla, David Jacobs, and Soheil Feizi. Low curvature activations reduce overfitting in adversarial training. *arXiv.org*, abs/2102.07861, 2021. 8

[83] Kihyuk Sohn, David Berthelot, C. Li, Zizhao Zhang, N. Carlini, E. D. Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv.org*, abs/2001.07685, 2020. 9

[84] Thilo Strauss, Markus Hanselmann, Andrej Junginger, and Holger Ulmer. Ensemble methods as a defense to adversarial perturbations against deep neural networks. *arXiv.org*, abs/1709.03423, 2017. 1

[85] David Stutz, Nandhini Chandramoorthy, Matthias Hein, and Bernt Schiele. Bit error robustness for energy-efficient dnn accelerators. In *MLSys*, 2021. 1, 7

[86] David Stutz, Matthias Hein, and Bernt Schiele. Confidence-calibrated adversarial training: Generalizing to unseen attacks. In *ICML*, 2020. 6

[87] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *arXiv.org*, abs/1710.08864, 2017. 1

[88] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 6

[89] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv.org*, abs/1312.6199, 2013. 1

[90] César Torres-Huitzil and Bernard Girau. Fault and error tolerance in neural networks: A review. *IEEE Access*, 5, 2017. 2

[91] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. *ICLR*, 2018. 1

[92] Yisen Wang, Difan Zou, Jinfeng Yi, James Bailey, Xingjun Ma, and Quanquan Gu. Improving adversarial robustness requires revisiting misclassified examples. In *ICLR*, 2020. 1, 7, 8

[93] Tsui-Wei Weng, Pu Zhao, Sijia Liu, Pin-Yu Chen, Xue Lin, and Luca Daniel. Towards certificated model robustness against weight perturbations. In *AAAI*, 2020. 1

[94] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 2018. 1

[95] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. *arXiv.org*, abs/2001.03994, 2020. 6

[96] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. In *NeurIPS*, 2020. 1, 5, 8

[97] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan L. Yuille. Mitigating adversarial effects through randomization. *ICLR*, 2018. 1

[98] Greg Yang, Tony Duan, Edward Hu, Hadi Salman, Ilya P. Razenshteyn, and Jerry Li. Randomized smoothing of all shapes and sizes. *arXiv.org*, abs/2002.08118, 2020. 1

[99] Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. Efficient defenses against adversarial attacks. In *AISec*, 2017. 1

[100] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Duane S. Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. *arXiv.org*, abs/1906.06316, 2019. 1

[101] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *NeurIPS*, pages 4944–4953, 2018. 1

[102] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically principled trade-off between robustness and accuracy. In *ICML*, 2019. 1, 8

[103] Jingfeng Zhang, Xilie Xu, Bo Han, Gang Niu, Li zhen Cui, Masashi Sugiyama, and Mohan Kankanhalli. Attacks which do not kill training make adversarial learning stronger. *arXiv.org*, abs/2002.11242, 2020. 9