The Best Defense is a Good Offense: Adversarial Augmentation against Adversarial Attacks (Supplementary)

Iuri Frosio, NVIDIA
ifrosio@nvidia.com

Jan Kautz, NVIDIA
jkautz@nvidia.com

1. Supplementary

1.1. Threat models and usage scenarios

The first row in Fig. 1 depicts a complete schematic representation of A^5, our framework for robust augmentation of both physical objects $\boldsymbol{w}$ or data $\boldsymbol{x}$, that makes them certifiably robust against MitM and physical (not tested here) adversarial attacks. The A^5 model includes an acquisition block, that can simulate image capturing with a camera or perform simple data normalization depending on the threat and training scenario. The acquired images $\boldsymbol{x}$ can be made certifiably robust against MitM attacks $\delta\boldsymbol{x}_A$ (crafted during their transmission to the classifier C) by a robustifier DNN R that runs on-the-fly within a protected environment inaccessible to the attacker (A^5/R and A^5/RC). We show also the use of A^5 for offline robustification of datasets (A^5/O) and of physical objects $\boldsymbol{w}$ (A^5/P and A^5/PC). The green blocks in Fig. 1 indicate the various A^5 recipes that are used to train R and C or to create robust physical objects or datasets.

A^5 can be used in a *white-box scenario* where the attacker has full access to the classifier C. However, since A^5 uses certified bounds, it is agnostic to the specific nature of the attack: we can certify protection against any MitM attack $\delta\boldsymbol{x}_A$, $||\delta\boldsymbol{x}_A|| < \epsilon_A$ in Fig. 1. In practice, the attacker can use any white-, grey- or black-box algorithm to generate the adversarial attack.

Notice that MitM attacks can be crafted while transmitting the dataset to the classifier C. In a first scenario (A^5/O in Fig. 1), we assume that A^5 runs offline, in a protected environment that is not accessible to the attacker, to preemptively robustify the samples of the dataset. This situation is however mostly of theoretical and little practical interest.

In a second scenario (A^5/R and A^5/RC in Fig. 1), we assume a robustifier R running on the acquisition device, in a protected environment; in this case robustification is performed on-the-fly, before transmitting the data to C (when data can be corrupted by a MitM attack) and without knowing the ground truth class. This case can find practical application in several fields including automotive [7] or anytime the capturing device communicates with another machine [11], under the assumption that R runs together with the acquisition device in a protected environment: the attacker may have full knowledge of R and C, but cannot modify $\boldsymbol{x}$ (before its transmission) or any internal state of the robustifier R.

In a third scenario, A^5 furnishes protection against MitM attacks by creating certifiably robust physical objects (A^5/P and A^5/PC in Fig. 1). In this case, the attacker may observe (but not interfere with) the object robustification process, that is performed offline, in a protected environment. This is in practice a very mild constraint, as robustification coincides with the physical creation of the objects. Practical examples are the design of certifiably robust road signs for automotive, fonts for OCR, or even audio signals [5, 6, 10].

Notice that, in its more general formulation (first row in Fig. 1), A^5 can also protects against physical adversarial attacks like adversarial patches [1] stick on top of a road sign, or adversarial sounds emitted in the environment to attack speech to text or voice recognition systems [5, 6, 10], although we do not test this case here.

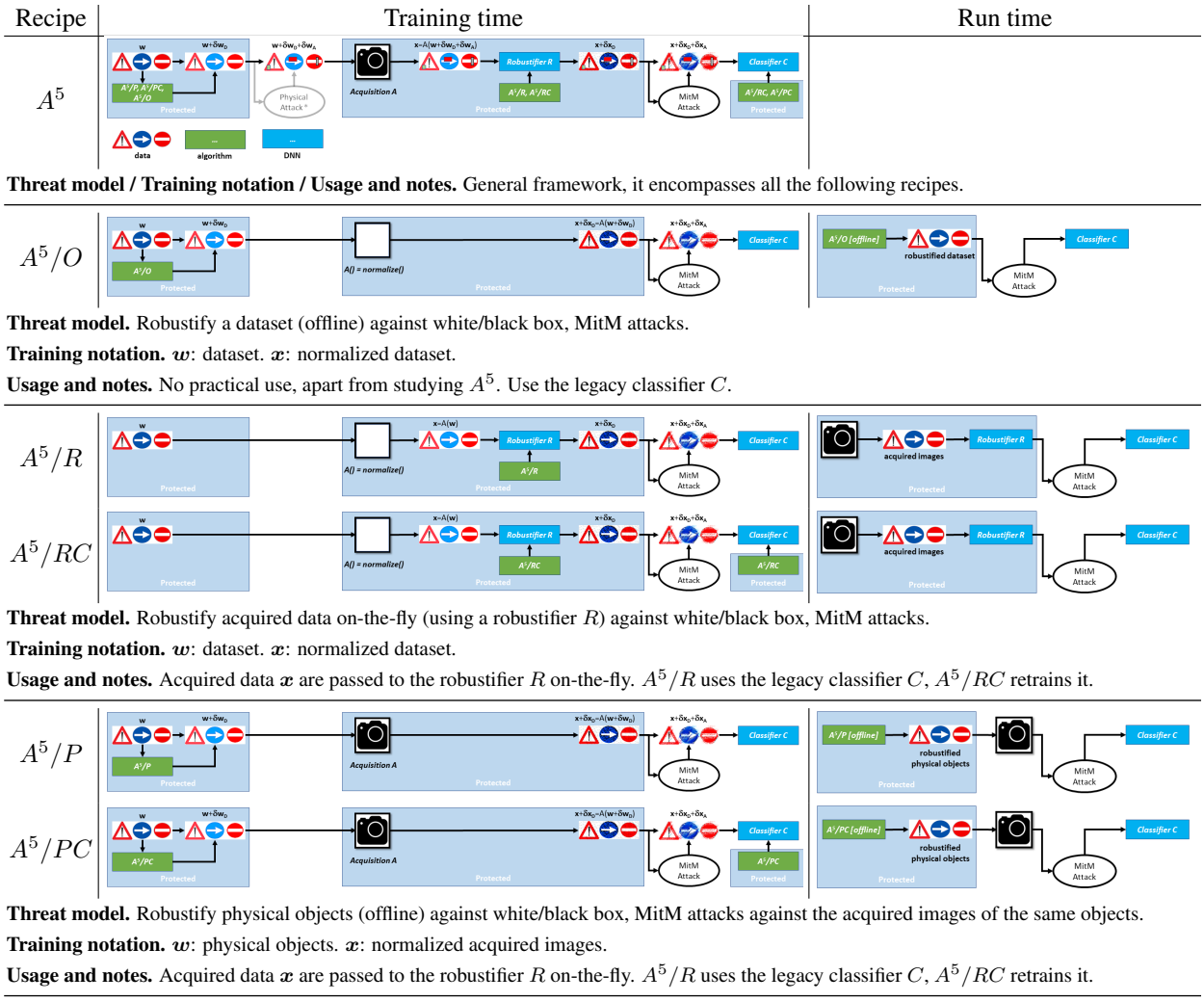
Beyond illustrating the typical use scenarios of A^5, Fig. 1 also serves as a reference for the notation used in the code that we share in https://github.com/NVlabs/A5.

1.2. Additional results

1.2.1 FashionMNIST

For completeness, we test A^5/R and A^5/RC on FashionMINST for $\epsilon_A = 0.1$, training with $\epsilon_A^C = 0.1$, $\epsilon_A^R = 0.1$ and $\epsilon_D = 0.3$. Table 1 reports the clean and certified errors, showing that A^5/R significantly boosts the clean and especially the certified errors over the CROWN-IBP baseline; A^5/RC reaches even better results.

Notice that A^5/RC and ℓ_∞-dist Net+MLP [12] achieve similar clean errors, while A^5 has better certified errors. This suggest that the integration of the design philosophy of ℓ_∞-dist Net (based on ℓ_∞-dist neurons that implement 1-Lipschitz functions) and A^5 may lead to even bigger improvements in terms of certified robustness in the future.



Threat model / Training notation / Usage and notes. General framework, it encompasses all the following recipes.

A^5/O

Threat model. Robustify a dataset (offline) against white/black box, MitM attacks.

Training notation. $\boldsymbol{w}$: dataset. $\boldsymbol{x}$: normalized dataset.

Usage and notes. No practical use, apart from studying A^5. Use the legacy classifier C.

A^5/R A^5/RC

Threat model. Robustify acquired data on-the-fly (using a robustifier R) against white/black box, MitM attacks.

Training notation. $\boldsymbol{w}$: dataset. $\boldsymbol{x}$: normalized dataset.

Usage and notes. Acquired data $\boldsymbol{x}$ are passed to the robustifier R on-the-fly. A^5/R uses the legacy classifier C, A^5/RC retrains it.

A^5/P A^5/PC

Threat model. Robustify physical objects (offline) against white/black box, MitM attacks against the acquired images of the same objects.

Training notation. $\boldsymbol{w}$: physical objects. $\boldsymbol{x}$: normalized acquired images.

Usage and notes. Acquired data $\boldsymbol{x}$ are passed to the robustifier R on-the-fly. A^5/R uses the legacy classifier C, A^5/RC retrains it.

Figure 1. The first row is a schematic representation of the full A^5 framework. It allows the robust augmentation of both physical objects $\boldsymbol{w}$ and acquired data $\boldsymbol{x}$ to make them certifiably robust against MitM and physical (not tested here) adversarial attacks. The remaining rows illustrate the A^5 recipes tested in our paper. The ground truth class is always known at training time, whereas it is unknown at run time.

Algo	Notes	Error [certified error]
CROWN-IBP [13]	From [12]	15.69% [21.99%]
CROWN-IBP [13]	Ours, $\epsilon_A^C = 0.1$	15.11% [22.92%]
ℓ_∞-dist Net+MLP [12]		12.09% [20.77%]
A^5/R	$\epsilon_A^R = 0.1, \epsilon_D = 0.3$	14.27% [15.75%]
A^5/RC	$\epsilon_A^R = 0.1, \epsilon_D = 0.3$	11.41% [15.53%]

Table 1. Error and certified errors on FashionMNIST, $\epsilon_A = 0.1$.

1.2.2 Tinyimagent

Large dataset are challenging for *all* the existing defenses. On TinyImageNet we got large improvements for A^5/O, smaller ones for A^5/RC (see Table 2) after 3 days of training, while the loss is still decreasing. We beat the state-of-the-art, but we believe that the classification landscape of C is not regular enough: A^5 exploits the gradient to find *ad hoc* solutions with class specific patterns (see Fig. 2, compare with the more regular defensive perturbations in Fig. 3 in the main paper). This is indicative of the fact that finding a general, regular defensive signal is hard for the robustifier R. More research is needed to handle large datasets and leverage A^5 at best.

1.3. Training recipes and network architectures

1.3.1 MNIST, classifier architecture

For the MNIST classifier C, we adopt a neural network with the *very large* architecture described in the IBP [4] and the CROWN-IBP [13] papers that is also implemented (and shared) in auto_LiRPA [8, 9].

Method	Error, $\epsilon_D = 16/255$
CROWN-IBP	75.33 [80.39 / 84.80]
A^5/O	18.14 [21.24 / 25.08]
A^5/RC	71.50 [75.19 / 78.54]

Table 2. Results on Tinyimagenet, for $\epsilon_A^C = 1/255$. We report the clean error [PGD error / verified error].

Figure 2. Robustification on tinyimagenet with A^5/O (first row) and A^5/RC, for $\epsilon_D = 16/255$. The left panel is the original image, the central one is robustified with the augmentation signal in the right panel.

1.3.2 MNIST, classifier training with CROWN-IBP

For training our CROWN-IBP classifiers, we follow the instructions reported in the CROWN-IBP [13] paper, apart from some minor details like the scheduling of the learning rate or the annealing of the training attack magnitude that we change as follows.

For training attack magnitudes $\epsilon_A^C < 0.3$, we train with batch size 100 for 100 epochs with an initial learning rate of 0.001, decreased at epochs 25 and 42 by a $10\times$ factor. We use a *SmoothedScheduler* (as implemented in auto_LiRPA [8, 9]) with starting epoch 3, length 18 epochs and $mid = 0.3$ to increase the value of ϵ_A^C during training. The worst case margins are computed in all cases using the mixed CROWN-IBP bounds [13]. For $\epsilon_A^C >= 0.3$, we use batch size 256 and 200 epochs, initial learning rate set to 0.0005 and decreased by a $10\times$ factor at epochs 130 and 190, and a *SmoothedScheduler* to increase ϵ_A^C starting at epoch 10, lasting for 50 epochs and with $mid = 0.3$. We use RMSprop as optimizer.

1.3.3 MNIST, robustifier architecture and A^5 training schedules

Our robustifier R for MNIST is a tiny convolutional network that takes a $1 \times 28 \times 28$ image in input, processes it with a convolutional layer with 32 3×3 filters with stride 1 and padding 1, a ReLU, and a second convolutional layer

$\epsilon_A^R = \epsilon_A$	ϵ_D	$\epsilon_A^C = 0.0$	$\epsilon_A^C = 0.1$	$\epsilon_A^C = 0.2$	$\epsilon_A^C = 0.3$	$\epsilon_A^C = 0.4$
0.1	0.05	88.65 [88.65, 88.67]	1.05 [1.88, 4.00]	1.08 [1.62, 2.28]	1.21 [1.93, 2.76]	1.58 [2.28, 2.88]
0.1	0.10	88.65 [88.65, 88.68]	1.07 [1.59, 3.20]	1.06 [1.44, 1.82]	1.03 [1.70, 2.24]	1.53 [2.13, 2.45]
0.1	0.20	88.65 [88.65, 88.66]	1.09 [1.43, 2.32]	1.01 [1.34, 1.64]	1.00 [1.44, 1.83]	1.27 [1.62, 2.06]
0.1	0.30	88.65 [88.65, 88.65]	1.01 [1.31, 2.10]	0.99 [1.23, 1.51]	1.03 [1.29, 1.59]	1.25 [1.53, 1.88]
0.1	0.40	88.65 [88.65, 88.65]	1.00 [1.30, 1.94]	0.98 [1.13, 1.44]	1.01 [1.24, 1.54]	1.30 [1.47, 1.93]
0.3	0.05	88.65 [88.65, 88.67]	3.43 [9.24, 12.41]	1.48 [5.77, 8.29]	1.63 [5.63, 7.78]	1.93 [5.53, 6.78]
0.3	0.10	88.65 [88.65, 88.66]	1.74 [5.44, 8.15]	1.29 [4.13, 6.46]	1.49 [4.23, 6.19]	2.09 [4.50, 5.32]
0.3	0.20	88.65 [88.65, 88.65]	0.98 [2.26, 5.78]	1.23 [2.34, 4.03]	1.39 [2.55, 3.96]	1.71 [3.03, 3.77]
0.3	0.30	88.65 [88.65, 88.66]	0.94 [1.49, 3.45]	1.11 [1.54, 2.44]	1.40 [2.18, 3.02]	1.56 [2.39, 2.98]
0.3	0.40	88.65 [88.65, 88.65]	0.94 [1.26, 2.41]	1.03 [1.27, 1.90]	1.15 [1.84, 2.70]	1.54 [2.02, 2.58]

Table 3. Error on clean data and (within brackets) under autoattack [2, 3] and verified, for MNIST, A^5/RC, under attack $\epsilon_A = \{0.1, 0.3\}$. During training we use $\epsilon_A^R = \epsilon_A$, and no data augmentation. Comparison with Table 5 in the main paper demonstrates the benefit of using data augmentation.

with 1 5×5 with stride 1 and padding 2 filter to produce the single channel, 28×28 output $\boldsymbol{z}$.

To train R with A^5/R on MNIST, given a pretrained (robust) classifier C, we use 50 epochs, batch size 256, initial learning rate 0.0005, whereas ϵ_A^R and ϵ_D are kept constant during the entire training process.

To fine tune R and C with A^5/RC on MNIST, given a pretrained (robust) classifier C and a pretrained roustifier R, we use 100 epochs and a fixed learning rate equal to 0.0005; ϵ_A^R and ϵ_D are kept constant during the entire training process. The batch size is 512 when data augmentation is used, 256 otherwise (see 1.3.4).

In all cases, we use RMSprop as optimizer.

1.3.4 Augmentation helps robustification by preventing overfitting

In the case of MNIST, A^5/RC, data augmentation (image random shifts by max 4 pixels and random rotation by max 10 degreees) leads to better robustification performances. Table 3 shows the clean, autoattack and certified errors for A^5/RC with no augmentation on MNIST; these metrics are clearly inferior to the ones reported in Table 5 in the main paper, showing the importance of using data augmentation.

Our interpretation of this result is that augmentation prevents overfitting, that is significant at least in case of MNIST where errors are small. This is further confirmed by the analysis of the training curves, where we observe the training error decreasing while validation goes up during training with A^5/RC without any data augmentation. The benefit of using data augmentation to train an MNIST robust classifier through CROWN-IBP or when using A^5/R is inferior or not present at all, probably because these algorithms achieve larger errors and overfitting is less likely.

1.3.5 CIFAR10, classifier architecture

For the CIFAR10 classifier C, we adopt a neural network with the *very large model* architecture described in the IBP [4] and the CROWN-IBP [13] papers and implemented in auto_LiRPA [8,9]. We do not train our own classifier with

CROWN-IBP — instead, we use the pretrained model distributed in https://github.com/huanzhang12/CROWN-IBP.

1.3.6 CIFAR10, robustifier architecture and A^5 training schedules

Our robustifier R for CIFAR10 is a small convolutional DNN with 3 convolutional layers with $64 \times 5 \times 5$ filters with stride 1, padding 2, $64 \times 5 \times 5$ filters with stride 1, padding 2, and $3 \times 5 \times 5$ filters with stride 1, padding 2, with ReLU activations between them. The robustifier takes a $3 \times 32 \times 32$ image in input and produces in output a vector $\boldsymbol{z}$ of the same size.

To train R on CIFAR10, given the aforementioned pretrained (robust) classifier C, we use 100 epochs, batch size 256, learning rate 0.0005, whereas ϵ_A^R and ϵ_D are kept constant during the entire training process.

To fine tune R and C with A^5/RC on CIFAR10, given a pretrained (robust) classifier C and a pretrained robustifier R, we use 200 epochs, batch size 512, learning rate 0.0005, whereas ϵ_A^R and ϵ_D are kept constant during the entire training process.

In both cases, we use the standard data augmentation procedure for CIFAR10 images (random horizontal flips, random crop by 4 pixels max, image normalization), and RMSprop as optimizer.

1.3.7 Additional notes for training with A^5/RC

Setting the right learning rate is fundamental for A^5/RC. If the learning rate is too small, the C/R pair do not move much away from the initial condition, falling into a local minimum of the cost function and achieving little gain with respect to the optimal R computed for the base C. If the learning rate is too large, training is unstable. With the right learning rate, we observe a small, initial decrease of the performances of the C/R pair, which we believe to be associated with exiting from the local minimum, but training later progresses towards much better clean and verified error rates. To train with A^5/RC, we also found important to use a sufficiently large batch size to guarantee the stability of the process.

Some of the results reported in the main paper suffer indeed from a sub-optimal choice of the learning rate (and potentially other training parameters). For instance, A^5/RC on CIFAR10 achieves far from optimal results for small values of ϵ_A^C. For simplicity (adoption of the same training parameters for all the metrics in the same Table) we decided to leave these metrics in the Tables without fine tuning, since the configurations affected by this issue are not the optimal ones. Results can clearly improve after better training parameter tuning, but the presence of these metrics in the Table highlights the importance of a good choice of the training parameters.

1.3.8 FashionMNIST

For FashionMNIST, we use the same architectures and recipies used for C and R in MNIST.

1.3.9 Tinyimagenet

For Tinyimagenet, we start from the legacy classifier trained with CROWN-IBP that can be downloaded following the instructions provided in https://github.com/Verified-Intelligence/auto_LiRPA/blob/master/doc/src/examples.md#certified-adversarial-defense-on-downscaled-imagenet-and-tinyimagenet-with-loss-fusion. To train the robustifier R and fine tune the classifier C, we use 1500 epochs, batch size 16, initial learning rate 0.0001 multiplied by a $0.8\times$ factor at epochs 500, 750, 1000, and 1250, $\epsilon_A^R = 1.1/255$ and $\epsilon_D = 16/255$.

1.3.10 Optical Character Recognition, classifier architecture

For OCR, we use a classifier C with 4 convolutional layers ($64 \times 5 \times 5$ filters with stride 2, padding 2, $32 \times 5 \times 5$ filters with stride 2, padding 2, $16 \times 3 \times 3$ filters with stride 2, padding 1, $8 \times 3 \times 3$ filters with stride 2, padding 1) with ReLU activations, followed by 1 fully connected layer with 512 features in input and output, ReLU activations and a last fully connected layer with 512 features in input and 62 in output. The input of C is a $3 \times 128 \times 128$ image, whereas the output is the vector of 62 logit values.

1.3.11 Optical Character Recognition, classifier training with CROWN-IBP

To train a traditional, non robust classifier C, we perform 1666 update steps using RMSprop as optimizer. In each update step we process a batch with 672 characters (see Fig. 4 in the main paper). The learning rate is initially set to 0.001 and decreased by a factor $10\times$ at steps 833 and 1250. Training images $\boldsymbol{x}$ are obtained by simulating the acquisition of the (robust) characters (first row in Fig. 4 in the main paper) as $\boldsymbol{x} + \delta\boldsymbol{x}_D = A(\boldsymbol{w} + \delta\boldsymbol{w}_D)$[1], where A includes: a random crop of max 5 pixels; a random rotation of max 5 degrees; a random perspective distortion with max distortion scale 0.25 (accordingly to the pytorch implementation of *transforms.RandomPerspective*); adding white Gaussian noise with standard deviation in a random range from 0.2/255 to

[1]Notice that $\delta\boldsymbol{x}_D$ and $\delta\boldsymbol{w}_D$ are null when training a standard or a robust CROWN-IBP classifier.

25/255; a random blur with σ in the range $[0.01, 1.0]$ pixels; and color jittering (with brightness, contrast, saturation and hue parameters equal to 0.1, accordingly to the pytorch implementation of *transforms.ColorJitter*).

To train a robust classifier C, we initialize the weights of C with those of the previously trained non robust classifier, then we use CROWN-IBP for an additional 2500 training steps. Each training step uses a batch of size 128. We use a learning rate 0.0005 that is decreased by a factor $10\times$ after 1500 and 2000 steps, and increase ϵ_A^C from 0.0 to 0.2 in 2000 training steps, using a *SmoothedScheduler* (as implemented in auto_LiRPA [8, 9]) with $mid = 0.3$.

1.3.12 Optical Character Recognition, A^5 training schedules

To compute the robustified characters $\boldsymbol{w} + \delta\boldsymbol{w}_D$ wth A^5/P, we start from a classifier C trained with CROWN-IBP. We run A^5/P for 2500 epochs, where in each epoch we process 10 batches each of size 6. We use a learning rate 0.005 that is decreased by a factor $10\times$ after 1500 and 2000 epochs, whereas ϵ_A^C is fixed to 0.125 for the entire training process. We use $\epsilon_D = 1$ to allow the pixels to be changed from black to white (or viceversa).

To train with A^5/PC, we start from a classifier C trained with CROWN-IBP. We run A^5/P for 10500 epochs, where in each epoch we process 10 batches each of size 6. We use a learning rate 0.005 that is decreased by a factor $10\times$ after 6000 and 8000 epochs, whereas ϵ_A^C is fixed to 0.125 for the entire training process. We use $\epsilon_D = 1$ to allow the pixels to be changed from black to white (or viceversa).

As the reader may have noticed, the adoption of the correct training schedule is generally important to guarantee the convergence of A^5 to an effective solution.

1.4. Quantification of robustification on image quality

In many applications, the raw output of the acquisition device (and therefore its visual appearance) is not of interest for the final user — this is for instance the case of a classifier C employed in an automotive or robotic vision system. On the other hand, for those applications where the images in output from the acquisition device are potentially consumable by human observers, it is legit asking if and how image robustification through A^5/O, A^5/R, and A^5/RC affects the image quality. A^5 is already created by design such that the L_∞ norm of the defensive perturbation does not exceed ϵ_D (*i.e.*, $||\delta\boldsymbol{x}_D||_\infty < \epsilon_D$), thus the user can modulate the worst case image degradation by setting ϵ_D. However, we do not know how deeply A^5/O, A^5/R, and A^5/RC leverage the available perturbation space while computing $\delta\boldsymbol{x}_D$. To quantify it, we measure the average Peak Signal to Noise Ratio (PSNR) between the original $\boldsymbol{x}$ and the robustified $\boldsymbol{x} + \delta\boldsymbol{x}_D$ image for the CIFAR10 experiment reported in Table 2 in the main paper. The PSNR values are reported in Table 4.

Algo	ϵ_D	Worst case	$\epsilon_A^R = 4/255$	$\epsilon_A^R = 8/255$	$\epsilon_A^R = 16/255$
A^5/O	4/255	36.09dB	47.04	46.40	44.88
	8/255	30.07dB	40.74	39.95	38.51
	16/255	24.05dB	34.56	33.90	32.26
	32/255	18.03dB	28.29	27.74	26.12
A^5/R	4/255	36.09dB	36.27	36.27	36.35
	8/255	30.07dB	30.31	30.30	30.35
	16/255	24.05dB	24.47	24.40	24.45
	32/255	18.03dB	18.75	18.70	18.61
A^5/RC	4/255	36.09dB	36.27	36.27	36.35
	8/255	30.07dB	30.27	30.27	30.35
	16/255	24.05dB	24.34	24.34	24.34
	32/255	18.03dB	18.57	18.56	18.52

Table 4. Average PSNR of the robustified $\boldsymbol{x} + \delta\boldsymbol{x}_D$ with respect to the vanilla $\boldsymbol{x}$ for CIFAR10 images and robustification with A^5/O, A^5/R, and A^5/RC and different settings (the same used for Table 2 in the main paper). The PSNR reported in the third column represents the worst case, where each pixel value is either increased or decreased by ϵ_D exactly.

This Table highlights a difference in the images robustified by A^5/O and those output by A^5/R and A^5/RC. In this first case, the image degradation is proportional to ϵ_D, as expected, but it is far from the worst case. In other words, A^5/O identifies local optima where many pixel values are changed by less than ϵ_D. On the other hand, both A^5/R and A^5/RC are characterized by PSNR values that are close to the worst case ones, meaning that the pixel values are often changed by either $+\epsilon_D$ or $-\epsilon_D$. Such observation is consistent with the images shown in Fig. 3 in the main paper, where $\delta\boldsymbol{x}_D$ shows a sparse structure for A^5/O, while it is characterized by large, constant value areas in case of A^5/R and A^5/RC.

Overall, A^5/RC provides a more effective robustification than A^5/O; this is achieved at the cost of a larger image degradation, anyway controlled by ϵ_D.

1.5. Resources

For training and testing A^5 we use either an HP Z8 G4 Workstation equipped with an Intel Xeon Gold 6128 @ 3.40GHz CPUs, RAM 48G and two NVIDIA GeForce 2080 Ti GPUs, each with 12G RAM, or an NVIDIA DGX-1 equipped with Intel Xeon E5-2698 v4 @ 2.20GHz CPUs, RAM 48G, and 8 NVIDIA Tesla V100-SXM2 GPUs, each with 32G RAM. All training and testing are done using a single GPU, running multiple experiments in parallel on the same machine when possible. Typical training times are in the order of few hours to for robust CROWN-IBP classifiers on MNIST, CIFAR10, FashionMNIST; A^5/O, A^5/R and A^5/RC take up to one day of training on the same data, while training on Tinyimagenet requires multiple days. Typical training times for A^5/P and A^5/PC on

the 62 characters dataset are again in the order of few hours.

References

[1] Tom Brown, Dandelion Mane, Aurko Roy, Martin Abadi, and Justin Gilmer. Adversarial patch. 2017. 1

[2] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, 2020. 3

[3] Francesco Croce and Matthias Hein. Mind the box: l_1-apgd for sparse adversarial attacks on image classifiers. In *ICML*, 2021. 3

[4] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models, 2018. 2, 3

[5] Sonal Joshi, Jesús Villalba, Piotr Żelasko, Laureano Moro-Velázquez, and Najim Dehak. Study of pre-processing defenses against adversarial attacks on state-of-the-art speaker recognition systems. *IEEE Transactions on Information Forensics and Security*, 16:4811–4826, 2021. 1

[6] Yao Qin, Nicholas Carlini, Ian J. Goodfellow, G. Cottrell, and Colin Raffel. Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. *ArXiv*, abs/1903.10346, 2019. 1

[7] Derui Wang, Chaoran Li, Sheng Wen, Surya Nepal, and Yang Xiang. Man-in-the-middle attacks against machine learning classifiers via malicious generative models. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2074–2087, sep 2021. 1

[8] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *arXiv: Learning*, 2020. 2, 3, 5

[9] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021. 2, 3, 5

[10] Hiromu Yakura and Jun Sakuma. Robust audio adversarial example for a physical attack. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5334–5341. International Joint Conferences on Artificial Intelligence Organization, 7 2019. 1

[11] Jianfei Yang, Han Zou, and Lihua Xie. Robustsense: Defending adversarial attack for secure device-free human activity recognition. *ArXiv*, abs/2204.01560, 2022. 1

[12] Bohang Zhang, Tianle Cai, Zhou Lu, Di He, and Liwei Wang. Towards certifying l-infinity robustness using neural networks with l-inf-dist neurons. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12368–12379. PMLR, 18–24 Jul 2021. 1, 2

[13] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020. 2, 3