7. Appendix

7.1. Experiments.

Further details regarding the implementation:

Details on the models used. All models on ImageNet are taken as pretrained versions from the torchvision[4] python package. For CIFAR10, both CNN7[5] as well as WResNet[6] are available on GitHub as pretrained versions. The CNN4 model is a standard convolutional network with layers of 32, 32, 64 and 64 channels, each using 3×3 filters and each layer being followed by a ReLU nonlinearity and 2×2 MaxPooling. The final layer is fully connected.

Training procedures. We used pretrained versions of all models except CNN4, which we trained for 50 epochs with RMSProp and a learning rate of 0.0001. For adversarial training, the models were trained for 50, 100 and 150 epochs using mixed batches of clean and corresponding adversarial (PGD) samples, matching the respective training schedule and optimizer settings of the clean models. We report results for the best performing variant. The exception to this is the WResNet model, for which an adversarially trained version was already available.

Setting the thresholds. The thresholds $\tau_{y,z}$ are set such that our statistical test achieves the highest possible detection rate (aka True Positive Rate) at a prespecified False Positive Rate of less than 1% (5% for Sections 5.6 and 5.7), computed on a hold-out set of natural and adversarially perturbed samples.

Determining attack strengths. For the adversarial attacks we consider, we can choose multiple parameters to influence the strength of the attack. Usually, as attack strength increases, at some point there is a sharp increase in the fraction of samples in the dataset where the attack is successful. We chose our attack strength such that it is the lowest value after this increase, which means that it is the lowest value such that the attack is able to successfully attack most of the datapoints. Note that weaker attacks generate adversarial samples that are closer to the original samples, which makes them harder to detect than excessively strong attacks.

Noise sources. Adding noise provides a non-atomic view, probing the classifiers output in an entire neighborhood around the input. In practice we sample noise from a mixture of different sources: Uniform, Bernoulli and Gaussian noise with different magnitudes. The magnitudes are sampled from a log-scale. For each noise source and magnitude, we draw 256 samples as a base for noisy versions of the incoming datapoints, although we have not observed a large drop in performance using only the single best combination of noise source and magnitude and using less samples, which speeds up the wall time used to classify a single sample by an order of magnitude. For detection, we test the sample in question against the distribution of each noise source, then we take a majority vote as to whether the sample should be classified as adversarial.

Plots. All plots containing shaded areas have been repeated over the dataset. In these plots, the line indicates the mean measurement and the shaded area represents one standard deviation around the mean.

Wall time performance. Since for each incoming sample at test time, we have to forward propagate a batch of N noisy versions through the model, the time it takes to classify a sample in a robust manner using our method scales linearly with N compared to the same model undefended. The rest of our method has negligible overhead. At training time, we essentially have to do perform the same operation over the training dataset, which, depending on its size and the number of desired noise sources, can take a while. For a given model and dataset, this has to be performed only once however and the computed statistics can then be stored.

7.2. Logistic classifier for reclassification.

Instead of selecting class z according to Eq. (6), we found that training a simple logistic classifier that gets as input all the $K-1$ Z-scores $\bar{g}_{y,z}(x)$ for $z \in \{1,..,K\}\backslash y$ can further improve classification accuracy, especially in cases where several Z-scores are comparably far above the threshold. Specifically, for each class label y, we train a separate logsitic regression classifier C_y such that if a sample x of predicted class y is detected as adversarial, we obtain the corrected class label as $z = C_y(x)$. These classifiers are trained on the same training data that is used to collect the statistics for detection. Two points are worth noting: First, as the classifiers are trained using adversarial samples from a particular adversarial attack model, they might not be valid for adversarial samples from other attack models. However, we confirm experimentally that our classifiers (trained using PGD) do generalize well to other attacks. Second, building a classifier in order to protect a classifier might seem tautological, because this metaclassifier could now become the target of an adversarial attack itself. However, this does not apply in our case, as the inputs to the metaclassifier are (i) low-dimensional (there are just $K-1$ weight-difference alignments for any given sample), (ii) based on sampled noise and therefore random variables and (iii) the classifier itself is shallow. All of these make it much harder to specifically attack the corrected classifier. In Section 5.7 we show that our method performs reasonably well even if the adversary is fully aware (has perfect knowledge) of the defense.

[4] https://github.com/pytorch/vision

[5] https://github.com/aaron-xichen/pytorch-playground

[6] https://github.com/MadryLab/cifar10_challenge

7.3. Additional results mentioned in the main text.

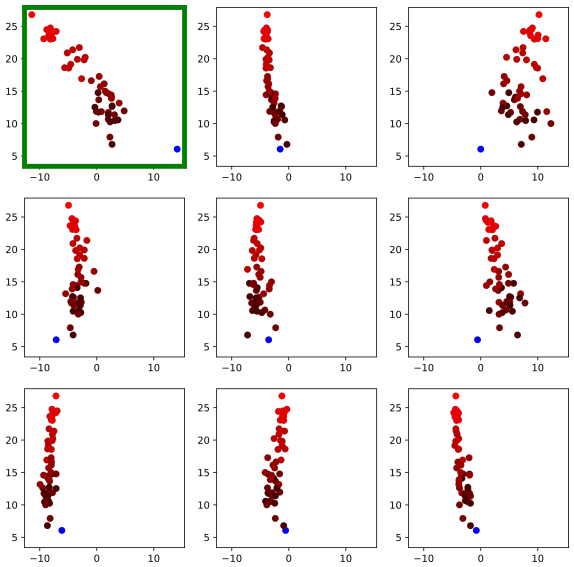

Figure 7. Noise-induced change of logit scores f_y (on the vertical axis) and f_z (on the horizontal axis). Different plots correspond to different classes $z \in \{1, .., K\} \backslash y$. The light red dot shows an adversarially perturbed example x without noise. The other red dots show the adversarially perturbed example with added noise. Color shades reflect noise magnitude: light = small, dark = large magnitude. The light blue dot indicates the corresponding natural example without noise. The candidate class z in the upper-left corner is selected. See Figure 1 for an explanation.

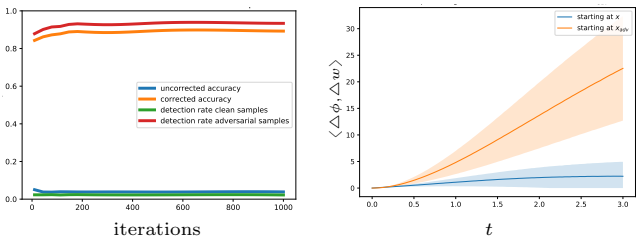

Figure 8. (Left) Detection rates and accuracies vs. number of PGD iterations. (Right) Noise-induced weight-difference alignment along $x^* + t\eta$ and $x + t\eta$ respectively. For the adversarial example, the alignment with the weight-difference vector between the true and adversarial class is shown. For the natural example, the largest alignment with any weight-difference vector is shown.

Table 7. Proximity to nearest neighbor. The table shows the ratio of the 'distance between the adversarial and the corresponding unperturbed example' to the 'distance between the adversarial example and the nearest other neighbor (in either training or test set)', i.e. $||x - x^*||_2 / ||x - x^{\text{nn}}||_2$.

PGD	$\epsilon_\infty = 2$	$\epsilon_\infty = 4$	$\epsilon_\infty = 8$
L^∞	0.021 ± 0.005	0.039 ± 0.010	0.075 ± 0.018
L^2	0.023 ± 0.006	0.043 ± 0.012	0.088 ± 0.019

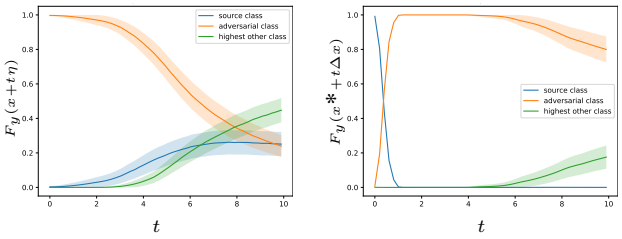

Figure 9. (Left) Softmax predictions $F_y(x + t\eta)$ when adding random noise to the adversarial example. (Right) Softmax predictions $F_y(x^* + t\Delta x)$ along the ray from natural to adversarial example and beyond. For the untargeted attack shown here, the probability of the source class stays low, even at $t = 10$.

7.4. ROC Curves.

Figure 10 shows how our method performs against a PGD attack under different settings of thresholds τ.

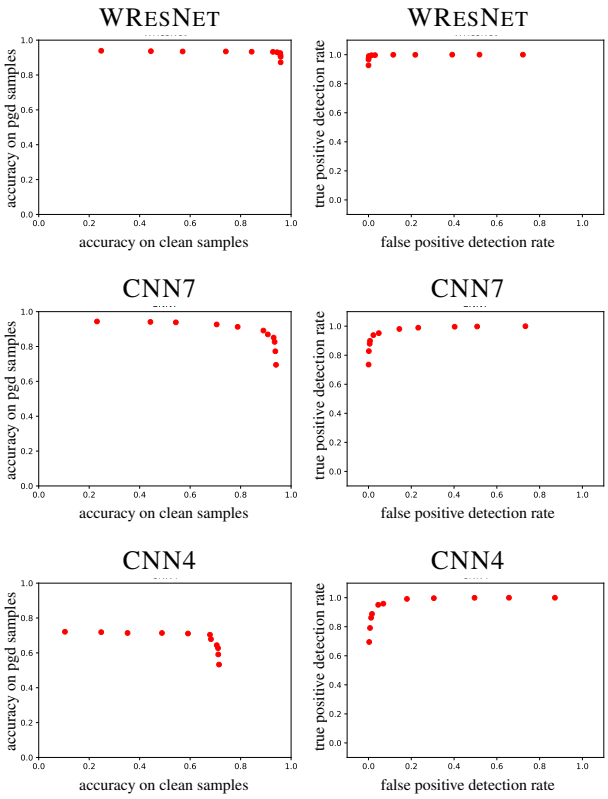

Figure 10. ROC-curves. Test set accuracies and detection rates on clean and PGD-perturbed samples for a range of thresholds τ on CIFAR10.