Breaking Transferability of Adversarial Samples with Randomness

Yan Zhou
University of Texas at Dallas
Richardson, Texas
yan.zhou2@utdallas.edu

Murat Kantarcioglu
University of Texas at Dallas
Richardson, Texas
muratk@utdallas.edu

Bowei Xi
Purdue University
West Lafayette, Indiana
xbw@stat.purdue.edu

ABSTRACT

We investigate the role of transferability of adversarial attacks in the observed vulnerabilities of Deep Neural Networks (DNNs). We demonstrate that introducing randomness to the DNN models is sufficient to defeat adversarial attacks, given that the adversary does not have an unlimited attack budget. Instead of making one specific DNN model robust to perfect knowledge attacks (a.k.a, white box attacks), creating randomness within an army of DNNs completely eliminates the possibility of perfect knowledge acquisition, resulting in a significantly more robust DNN ensemble against the strongest form of attacks. We also show that when the adversary has an unlimited budget of data perturbation, all defensive techniques would eventually break down as the budget increases. Therefore, it is important to understand the game saddle point where the adversary would not further pursue this endeavor.

Furthermore, we explore the relationship between attack severity and decision boundary robustness in the version space. We empirically demonstrate that by simply adding a small Gaussian random noise to the learned weights, a DNN model can increase its resilience to adversarial attacks by as much as 74.2%. More importantly, we show that by randomly activating/revealing a model from a pool of pre-trained DNNs at each query request, we can put a tremendous strain on the adversary's attack strategies. We compare our randomization techniques to the Ensemble Adversarial Training technique and show that our randomization techniques are superior under different attack budget constraints.

KEYWORDS

adversarial attacks; adversarial machine learning; deep neural networks

1 INTRODUCTION

Vulnerabilities of machine learning algorithms have been brought to attention of researchers since the early days of machine learning research [2, 7, 24]. Adversarial machine learning as a general topic has been extensively studied in the past, covering a wide variety of machine learning algorithms [3, 6, 13, 14, 17, 23, 29, 45]. However, it is the recent popularity of Deep Neural Network (DNN) that has put the adversarial learning problem under the spotlight across a large number of application domains, especially in image classification and cybersecurity. The ease of breaking down DNNs by perturbing as little as a few pixels in an image raised great concerns on the reliability of DNNs, especially in application domains where active adversaries naturally exist. The seemingly extreme vulnerability of DNNs has sparked booming academic development in robust DNN research. Two major directions dominate the current effort in searching for a robust remedy, both addressing the sensitive nature of DNNs. One is to manipulate the source input such that the hidden "absurdity" in its adversarial counterpart would disappear after the preprocessing [20]; the other is to directly address the root cause of the sensitivity of DNNs [31]. So far none has proved to be the consistent winner.

Perhaps, the most disturbing thought about adversarial attacks is the contagion effect of the crafted adversarial samples, formally described as *transferability* [39]. It has been shown that adversarial samples that defeat one DNN model can easily defeat other DNN models. Furthermore, adversarial samples computed from a DNN model may successfully defeat other types of differentiable and non-differentiable learning techniques including support vector machines, decision trees, logistic regression, and even the ensemble of these algorithms [32]. The idea of transferability has been widely, and undoubtedly, accepted as a common phenomenon, leading research efforts away from further investigating the nature of transferability. In fact, the success story of black-box attacks is strictly based on the assumption of transferability [9]. To attack a black-box learning model, of which the internal training data and learned parameters are kept secret from the attackers, a surrogate learning model is built by probing the black-box model with unlabeled samples. Once the label information is obtained for the unlabeled samples, the surrogate learning model will be trained, and adversarial samples against the surrogate model can be computed thereafter. Based on the idea of transferability, these adversarial samples are believed to be able to defeat the black-box model as well.

In this paper, we argue that transferability should not be automatically assumed. Instead, it has a strong dependence on the depth of invasion that the adversarial samples are allowed in the input space. We show that by simply adding small random noises to the learned weights of a DNN model, we can significantly improve the classification accuracy by as much as 74.2%, on condition that the adversary's goal is to misclassify a sample with minimum perturbation cost. On the other hand, if the adversary can modify samples without cost constraints, that is, they can arbitrarily modify samples until they are misclassified without penalty, no robust learning algorithm would exist to defeat this type of attack. Therefore, transferability is essentially the observable trait of the severity of adversarial data perturbation. In addition, we show that by *randomly selecting a DNN model from a pool of DNNs at each query request*, including probing request and classification request, we can effectively mitigate transferability based adversarial attacks. Furthermore, we show that the ensemble of a set of DNN models can achieve nearly perfect defense against adversarial attacks when the attack budget is low. Even when the attack budget is high, the ensemble is significantly more robust than the *Ensemble Adversarial Training* technique [40].

Our contributions can be summarized as follows:

- We investigate transferability of adversarial samples in terms of decision boundary-robustness and the intensity of adversarial attacks.
- We demonstrate that transferability should not be automatically assumed for a learning task, and by holding on to a subset F of DNN models, we can defeat the adversary who has the perfect knowledge of a DNN model built with the same training data as F, but with different initialization points for stochastic gradient descent. Furthermore, we demonstrate that the ensemble of F can provide the strongest defense against severe attacks.
- We demonstrate that by adding a small Gaussian random noise to the weights of a pre-trained DNN, we can significantly increase the robustness of the DNN model.
- We compare our techniques to the existing ensemble adversarial training technique and show that our proposed randomization approaches are more robust against the state-of-the-art attacks.

2 RELATED WORK

Szegedy et al. [39] showed that adversarial samples computed from one DNN model can effectively defeat the same DNN model, achieving 100% attack success rate or 0% accuracy on the adversarial samples. In addition, the adversarial samples computed from one DNN model are general enough to defeat other DNN models trained with different architectures, or with different training samples. Papernot et al. [33] presented a black-box attack strategy that involves training a local model on inputs synthetically generated by an adversary and labeled by a black-box DNN. They compute adversarial examples from the local substitute model, and successfully defeat the black-box DNN. Although they achieved very impressive attack success rates, their targets (the black-box DNNs) are online deep learning API that remain static (no learning weights variations) after been trained. They also presented evidence of transferability of adversarial samples across several DNN models trained with different layer parameters. Liu et al. [28] presented a study on transferability of both targeted and untargeted adversarial examples, where targeted adversarial samples are computed to be misclassified into the classes the adversary desires, while untargeted adversarial samples do not have desired labels specified. Tremèr et al. [40] discovered that if two models share a significant fraction of their subspaces, they enable transferability. They showed that adversarial samples computed from linear models can transfer to higher order models. They also presented a counter example where transferability between linear and quadratic models failed. Goodfellow et al. [18, 33] showed that explicit defenses such as distillation [34] and adversarial training [18] are not effective on eliminating transferability of adversarial samples.

Defensive strategies are quickly developed in response to the observed vulnerabilities of DNN models. A typical approach is to enhance a DNN model by re-training on its own adversarial samples [18]. The problem with this type of approach is that it may overfit to the set of adversarial samples in the training set. In addition, once the adversary knows the defense strategy, it may compute a new set of adversarial samples, optimized against the newly trained DNN model. Other defense techniques [4, 35, 44], including fine-tuning, distillation, and stability training, are not always effective against adversarial samples [9]. Chalupka et al. [11] presented a learning procedure for training a manipulator function that is robust against adversarial examples. However, the technique cannot obtain state-of-the-art accuracy on the non-attacked test sets [4]. More recently, several new defenses were proposed [8, 15, 20, 30, 31, 36, 37, 43]. However, all except one failed the latest attacks [1]. The most promising one was presented by Madry et al. [31]. They presented a minimax style solution to L_∞ attacks under the traditional robust optimization framework [38, 41, 42]. The inner maximization finds the attacks that maximize the loss, and the outer minimization finds the learning parameters that minimize the expected loss. Although the problem can be approximately solved, it is difficult to perform its adversarial retraining on large scale image datasets. In addition, its robustness to attacks with other distance metrics is limited.

Besides defense techniques, many detection techniques were proposed to differentiate adversarial samples from benign ones. Carlini and Wagner [9] specifically studied ten detection schemes, including principle component detection [5, 21, 27], distributional detection [16, 19], and normalization detection [16, 19, 27]. Among all the detection techniques, only randomization with dropout showed promising results.

3 ROBUST LEARNING WITH RANDOMIZATION

We consider the adversarial learning problem where adversarial data perturbation is bounded by $\epsilon \geq 0$. If the attack is unbounded, the adversary could simply change every feature in a sample and make it identical to the target samples. There does not exist a successful defense for this type of attack. A more surreptitious attack scenario is where perturbations are sufficiently small such that humans can make the right classification, but strong enough to foil a trained DNN model.

3.1 Transferability v.s. Attack Intensity

Given a DNN model f trained on the training set $\mathcal{T}$, let $x_* \in \mathcal{D}$ be a boundary data point of f, that is, x_* sits right on the decision boundary of f and the decision for x_* is undefined. We define the robustness of the DNN model f at x_* as the maximum perturbation $\rho(f, x_*)$ to x_* such that for every $x \in \mathcal{T}$, $f(x + \rho(f, x_*)) = f(x)$. In other words, $\rho(f, x_*)$ is the minimum perturbation any data point in the training set has to have in order to reach x_* on the decision boundary. Therefore, the robustness of the DNN model f can be defined, in terms of points on the decision boundary, as:

$$\mu(f) = \mathbb{E}_{x_*}\{\rho(f, x_*) | f(x_*) = \varnothing\},$$

where $f(x_*) = \varnothing$ indicates the decision for x_* is undefined on the boundary. Higher $\mu(f)$ values suggest greater expected minimum distance to travel in order to step across the decision boundary, and thus more robust f against adversarial data perturbation. We refer to $\mu(f)$ as *boundary robustness*.

The transferability of an adversarial perturbation to a data point $x \in \mathcal{D}$ depends on the attack intensity, that is, the expected minimum perturbation $\mu(f, \epsilon)$, bounded by ϵ, to defeat the DNN model f. Attack intensity can be defined by considering point-wise robustness [4]:

$$\begin{aligned} \rho(f, x) &= \inf\{\epsilon\} \\ s.t. \quad & ||x' - x||_p \leq \epsilon, f(x') \neq f(x) \end{aligned}$$

that is, given an instance $x \in \mathcal{D}$, $\rho(f, x)$ is the minimum upper bound of the perturbation for which a DNN model f fails to classify correctly. $||\cdot||_p$ is the L_p norm, and $\epsilon \geq 0$ specifies the upper bound of the *attack intensity*:

$$\mu(f, \epsilon) = \mathbb{E}_{x \in \mathcal{D}}\{\rho(f, x) | \rho(f, x) \leq \epsilon\},$$

which is the expected bounded perturbation for which f is susceptible to adversarial attacks. If *boundary robustness* $\mu(f)$ is more likely to be weaker than the *attack intensity* $\mu(f, \epsilon)$, we consider the attack against f is generally transferable; otherwise, the attack is not transferable:

$$tr(f, x') = \begin{cases} 1 & \text{if } P[\mu(f) > \mu(f, \epsilon)] < \delta \\ 0 & \text{Otherwise} \end{cases}$$

where $0 < \delta < \frac{1}{2}$, tr is the transferability. Exhaustively searching the sample space to determine the expected minimum distance to the nearest adversarial sample is computational intractable. Instead, we can investigate the spread of the DNN model distribution in the version space. The greater the difference between the decision boundaries of a pair of DNN models in the version space, the more difficult for the adversary to defeat both models with a single adversarial perturbation, that is, a transferable perturbation. In the next section, we discuss how to measure the spread of the DNN model distribution in the version space, and present two randomization scheme to enhance the existing DNN models.

3.2 Robust DNN Learning

Let $F = \{f_{i|i \in (0,\infty)}\}$ be a set of DNN models trained with stochastic gradient descent (SGD) from random initialization points. We assume the weight of the DNN model $f \in F$ is a multivariate continuous random variable W, with a probability density function $\phi(w)$, mean μ, and covariance matrix Σ. A trained DNN model should satisfy:

$$\mathbb{P}_{w \sim \mathcal{N}(\mu, \Sigma)}(f(x) = y) > 1 - \gamma$$

for a given sample x with a true label y and $\gamma \in (\frac{1}{2}, 1)$.

We are interested in finding out whether a DNN model f_i is robust against adversarial samples X^j_{adv} computed from f_j where $i \neq j$. If the decision boundaries of f_i and f_j are quite different from each other, the adversary would inevitably have a much more difficult time defeating f_i with X^j_{adv} without knowing f_i exactly. To estimate the difference of the decision boundaries among the DNN models in F, we measure the spread of the weight W—a vector of n continuous random variables with mean μ and covariance matrix Σ.

We compute the differential entropy $h(W)$ to measure the spread of W with a pdf $\phi(w)$:

$$h(W) = \int_{-\infty}^{\infty} \phi(w) \log \phi(w) dw$$

According to the maximum entropy theorem [12],

$$h(W) \leq \frac{1}{2} \log[(2\pi e)^n |\Sigma|]$$

with equality if and only if $W \sim \mathcal{N}(\mu, \Sigma)$. A high differential entropy value implies high spread of W, and therefore the decision boundaries of any pair of models are largely different in the version space. As a result, attacks on one DNN model may not easily transfer to defeat other DNN models in the version space.

In an adversarial learning task, to make the learning model robust, two conditions must be satisfied: 1.) using different decision boundaries for two different types of queries: prediction and attack probing; and 2.) the expected distance between two decision boundaries is sufficiently large. More specifically, we need a run-time setup such that the adversary may have a perfect knowledge of one of our DNN models $f_a \in F$ for computing adversarial samples, but the DNN model f_p used for prediction is highly likely to be different from f_a. In addition, the more spread out the random weight vectors drawn from $\phi(w)$ for f_a and f_p are, the more robust f_p will be against adversarial samples X^a_{adv} computed from f_a. We discuss two randomization schemes for robust DNN learning below.

3.2.1 Random Selection of DNNs for Different Queries. We train a set of DNNs $F = \{f_{i|i \in [1,M]}\}$. At each query request, a DNN model $f_i \in F$ is randomly selected. If it is a query Q_a from an adversary, the DNN model is returned for the adversary to compute adversarial samples; if the query Q_p is from a user for predicting for sample x, the prediction made by $f_i \in F$ for x is returned. The process is shown in Figure 1.

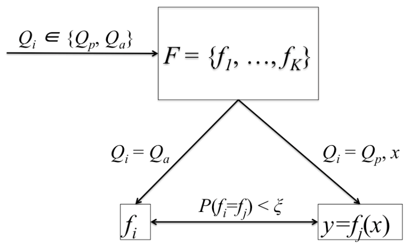

Figure 1: Random selection of DNNs f_i and f_j from F for different query types.

The attack is *partially white-box attack since the adversary has the perfect knowledge of one DNN model in the set.* If the total number of pre-trained DNNs $M > \frac{1}{\xi}$ for $0 < \xi < 1$, the probability $P(f_a = f_p | f_a, f_p \in F) < \xi$. Therefore, the more DNN models we train, the less likely that f_a is the same as f_p. If the spread of W is sufficiently large, f_p would be robust to attacks computed from f_a.

The robustness of this randomization scheme depends on whether we can keep a subset of the pre-trained DNNs secrete, and whether the decision boundaries of the pre-trained DNN models are largely different. In the case where there is a collision between f_a and f_p, we can make more robust predictions with the ensemble of the entire set, or the subset, of the pre-trained DNNs.

3.2.2 Randomizing a DNN with Random Weight Noise. In practice, training a large number of DNN models may not be feasible because of the sheer size of the problem. In our second randomization scheme, we simply introduce randomization to a DNN model by adding a small random noise to the weight of a pre-trained DNN, as shown in Figure 2. For the simplicity of exposition, we consider a two layer neural network:

$$f(x) = \arg\max_{\ell} w''g(w' \cdot x),$$

where g is the activation function, $w' \sim \mathcal{N}(\mu', \Sigma')$, $w'' \sim \mathcal{N}(\mu'', \Sigma'')$ are multivariate random weight vectors, with means μ' and μ'', covariance matrices Σ' and Σ''. We add a Gaussian random noise to the weight vectors w' and w'':

$$\begin{aligned} w' &= w' + \Delta w' \\ w'' &= w'' + \Delta w'' \end{aligned}$$

where $\Delta w' \sim \mathcal{N}(\mu', \nu\delta')$ and $\Delta w'' \sim \mathcal{N}(\mu'', \nu\delta'')$, δ' and δ'' are the variances of w' and w'', respectively. $0 < \nu < 1$ is a small constant used to compute the variances of the Gaussian random noise added to the weight vectors.

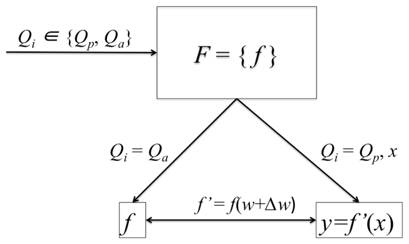

Figure 2: Adding random noise to a DNN model f to create a new model f' for making predictions.

This randomization scheme works if 1.) adding a small random noise to the weight does not affect the predictive accuracy on the original non-corrupted samples; and 2.) the new model is not only accurate but more resilient to attacks against the original model. The clear advantage of this technique is computational efficiency. We only need to train one model, and then add random noise to the model to transform it to a more robust model. Note that we can create a new DNN model by adding small random noise to the pre-trained one at each query request for prediction, with negligible cost.

4 EXPERIMENTS

We experiment on three datasets: MNIST, CIFAR-10, and German Traffic Sign [22, 25, 26]. The MNIST dataset has 60,000 training instances and 10,000 test instances. Each instance is a 28x28 image. There are ten digit classes for the classification task. The CIFAR-10 dataset has 60,000 images in 10 classes. There are 6000 images in each class. Each instance is a 32x32 color image. There are 50,000 images for training and 10,000 for testing. The German Traffic Sign dataset has 51,839 images in a total of 43 classes. Image sizes vary between 15x15 to 250x250 pixels. Each image is resized to 32x32. We used 39,209 for training, and 12,630 images for testing. We compare our randomness techniques to the *Ensemble Adversarial Training* technique [40].

The architecture of the DNN model for CIFAR-10 includes two convolution layers with 64 filters, two convolution layers with 128 filters, two fully connected layers, and one output layer. For MNIST, the DNN model consists of two convolution layers with 32 filters, two convolution layers with 64 filters, two fully connected layers, and one output layer. For the Traffic Sign dataset, the model includes two convolution layers with 64 filters, two convolution layers with 128 filters, two fully connected layers and one output layer. Details about layer component are given in Table 1.

As a reminder, we introduce randomness in two different ways: 1.) by randomly selecting a model from a large pool of DNNs at each query request, either for probing or for prediction; and 2.) by randomly adding a small noise to the weights of a trained DNN at each request. In both cases, we investigate the predictive accuracy of a single random DNN, and the predictive accuracy of an ensemble of 10, 20, and 50 random models, respectively. We add Gaussian random noise to the learned weights with zero mean and 0.01 standard deviation. We train the pool of DNNs using stochastic gradient descent from random initialization points.

We directly apply Carlini and Wagner's latest L_2 and L_∞ attacks [9, 31] in our experiments. We tested both targeted and untargeted attacks by specifying whether the adversary has a desired label to target. Depending on how closeness is measured, two different types of attacks are performed: L_2 attack (measuring the standard Euclidean distance) and L_∞ attack (measuring the maximum absolute change to any pixel). We randomly select 1000 samples from the test set, compute their adversarial samples, and test various DNN models on the adversarial samples. We repeat each experiment 10 times, and report the averaged results. For *Ensemble Adversarial Training* [40], we generate 10,000 adversarial samples computed from different DNNs, and add the 10,000 adversarial samples with the true labels to the training set to train a new DNN model.

4.1 L_2 Targeted Attacks

In this set of experiments, we investigate the transferability of adversarial samples generated using the L_2 attack model when the attack is targeted. We use Carlini and Wagner's iterative L_2 attack algorithm, which is superior to other existing attacks, to generate the adversarial samples [10]:

$$x^* = \arg\min_{x' \in \mathcal{D}} ||x' - x||_2^2 + c \cdot l(x'),$$

where $l(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -k)$, Z represents the logits of a DNN model f, and k controls the confidence of attack: when the confidence is zero, the attack just succeeds; when the confidence k is higher, the adversarial sample is more likely to be misclassified as the target. We increase the attack budget by setting the confidence value increasingly larger to simulate the attack intensity. The higher the confidence is, the more severe the attacks are. Figure 3 demonstrates how the confidence value affects the quality of the images after the attacks. The attack is already noticeable when the confidence value is zero, that is, when the attacker fools the DNN with minimum data perturbation. As the confidence value increases, the quality of the images drops significantly after the attacks.

Table 1: The architecture of the DNN models for CIFAR-10, MNIST, and German Traffic Sign datasets.

Dataset	Layer Component
CIFAR-10	1. two 2D convolution layers: 64 filters, stride 3; 2D Max Pooling: kernel size 2; ReLU activation 2. two 2D convolution layers: 128 filters, stride 3; 2D Max Pooling: kernel size 2; ReLU activation 3. two fully connected layers: 256 output; ReLU activation 4. one output layer: 10 output
MNIST	1. two 2D convolution layers: 32 filters, stride 3; 2D Max Pooling: kernel size 2; ReLU activation 2. two 2D convolution layers: 64 filters, stride 3; 2D Max Pooling: kernel size 2; ReLU activation 3. two fully connected layers: 200 output; ReLU activation 4. one output layer: 10 output
Traffic Sign	1. two 2D convolution layers: 64 filters, stride 3; 2D Max Pooling: kernel size 2; ReLU activation 2. two 2D convolution layers: 128 filters, stride 3; 2D Max Pooling: kernel size 2; ReLU activation 3. two fully connected layers: 256 output; ReLU activation 4. one output layer: 43 output

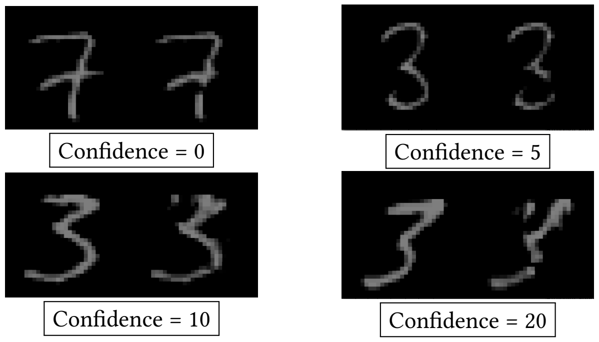

Figure 3: Quality of images after adversarial attacks with different confidence values.

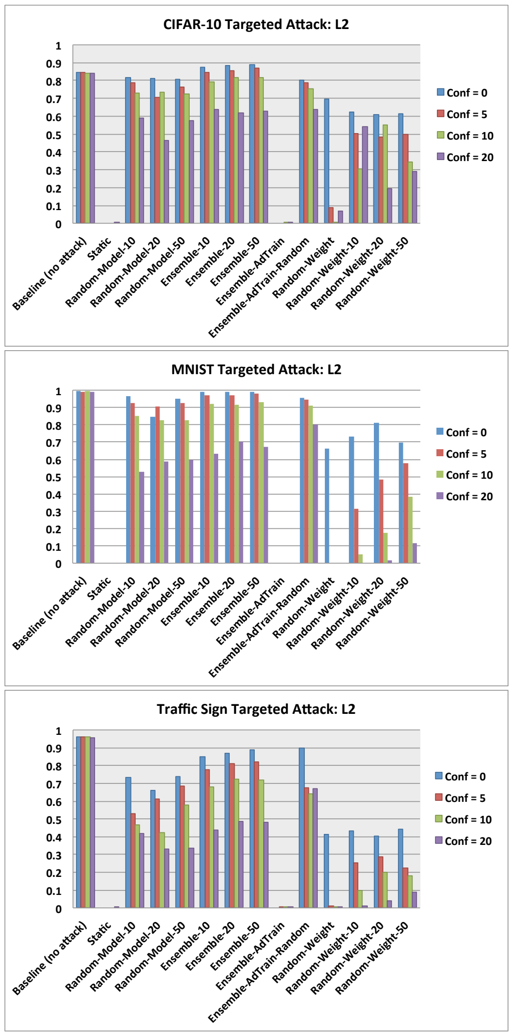

Figure 4: Results of targeted L_2 adversarial attacks on CIFAR-10, MNIST, and Traffic Sign datasets.

Figure 4 shows the detailed results on the three datasets. For each dataset, we compare the **Baseline** accuracy on the original data, the **Static** accuracy on perturbed data when the DNNs for attack probing and prediction are the same, the accuracy of our first randomization scheme **Random-Model-**n where there are $n = 10, 20, 50$ models in the pool, the accuracy of the ensemble of the pool of DNNs in our first randomization scheme **Ensemble-**n where $n = 10, 20, 50$, the accuracy of the *Ensemble Adversarial Training* technique [40] **Ensemble-AdTrain** and **Ensemble-AdTrain** combined with our first randomization scheme **Ensemble-AdTrain-Random**, and the accuracy of our second randomization scheme and its ensembles **Random-Weight** and **Random-Weight-**n where $n = 10, 20, 50$. Detailed explanations are given below.

4.1.1 Attacking and Predicting with A Static DNN. First, we investigate the influence of the adversarial samples on the same DNN model that has been provided to the attacker to compute the adversarial samples. In Figure 4 (and Tables 3, 4, and 5 in Appendix A), **Static** corresponds to the accuracy of the attacked DNN model on the adversarial samples when the confidence value $conf = 0, 5, 10, 20$, respectively. In all cases, the attacks succeeded with nearly 100% success rate, and the accuracy of the DNN model dropped significantly—*nearly 0% in all three cases and vanished in the plots*, compared to the **Baseline** accuracy on the non-attacked samples.

4.1.2 *Attacking and Predicting with A Randomly Selected DNN.*

We are interested in finding out what would happen if we hold on to some pre-trained DNN models. In other words, the attacker gets to randomly pick a model f_a from a pool of models to attack, and use it to generate the adversarial sample. We then randomly select a DNN f_p, with a probability $P(f_a = f_p) \leq \frac{1}{n}$ where n is the number of DNNs in the pool, to make the prediction for the adversarial sample. We set the pool size to 10, 20, and 50, respectively, shown as **Random-10**, **Random-20**, and **Random-50** in the figure (and the tables). It is clear that by keeping some pre-trained DNNs secrete, the adversary failed to fool the DNN that is randomly selected from the pool of DNNs on the CIFAR-10 and MNIST datasets. On the German Traffic Sign dataset, the attack success rate is greater, but not as devastating.

4.1.3 *Attacking A Randomly Selected DNN and Predicting with An Ensemble of DNNs.*

Naturally, the next question is what if we use the ensemble of the pool. Again, the adversary has random access to one of the DNN models in the pool to generate the adversarial sample, but we are going to use the majority vote of the DNN models to make the final prediction for the adversarial sample. The results are shown in the figure as **Ensemble-10**, **Ensemble-20**, and **Ensemble-50**, respectively. The ensemble approach has even better success in mitigating adversarial attacks. In some cases, it has better accuracy than the **Baseline**. Even on the German Traffic Sign dataset, the ensemble achieved significantly better accuracy than the single random DNN model.

4.1.4 *Comparing to Ensemble Adversarial Training.*

Now the question is: can we do better than *Ensemble Adversarial Training*? The results of *Ensemble Adversarial Training* are shown as **Ensemble-AdTrain** in the figure, and they are as poor as the **Static** DNN, with nearly 0% accuracy across all datasets under different confidence values. This makes sense as the adversarial learning process is essentially an arms race between the defender and the attacker. As long as the attacker has the learned model, he/she can apply the same attack strategy all over again, and defeat the new model. We again demonstrate the power of randomness by setting a pool of 10 **Ensemble-AdTrain** DNNs, and repeat the experiments as described in the previous case, and the results are nearly as good as those of our random approach. On the German Traffic Sign, **Ensemble-AdTrain** even outperformed our random approach.

4.1.5 *Attacking A DNN and Predicting with the Random-Weight DNN.*

Our second randomization approach may sound naïve at first. In this approach, we simply take a trained DNN model, and then add a small Gaussian random noise to its weights, and use it to make predictions for the adversarial samples. The results, shown as **Random-Weight** in the figure, are surprisingly strong when the attacker is not aggressive, that is, when the confidence value is zero. By adding small random noise to the trained DNN, we improve the accuracy by as much as 69.7% on the CIFAR-10 dataset, 66.3% on the MNIST dataset, and 41.4% on the Traffic Sign dataset, compared to the **Static** DNN.

4.1.6 *Attacking A DNN and Predicting with the Ensemble of Random-Weight DNNs.*

The problem with the **Random-Weight** DNN is that when the attack intensifies with higher confidence values, it immediately degraded to the **Static** DNN. Naturally, we consider the ensemble of a set of **Random-Weight** DNNs. Note that generating a pool of **Random-Weight** DNNs is very cheap, involving no re-training. The results are shown as **Random-Weight-10**, **Random-Weight-20**, and **Random-Weight-50** in the figure. The ensemble has significant improvement on the MNIST dataset when the confidence is zero, but not so much on the other two datasets. However, the ensemble appears to be significantly more resilient to attacks when the confidence value gets larger.

When the attack intensity increases, the strength of our random approaches decreases, but the ensemble can slow down the deterioration. Nevertheless, transferability should not be automatically assumed, and there is a good chance that a simple workaround such as adding random noise to the trained model would alleviate the problem.

4.2 L_2 Untargeted Attacks

In this experiment, we consider an adversary who has no desired target class and his/her goal is solely to get a sample misclassified. Figure 5 shows the results on all three datasets.

Similar to the case of targeted attacks presented in the previous section, the untargeted attacks are also very successful when the adversary has access to the DNN model, shown as **Static** in Figure 5 and Tables 6, 7, and 8 in Appendix A. The ensemble adversarial training technique, shown as **Ensemble-AdTrain**, also has no or very little resilience to untargeted attacks.

Both of our randomization techniques significantly improve the robustness of the DNN models.

- The ensembles of the DNNs in our first randomization technique **Ensemble-*n*** with $n = 10, 20, 50$ are the strongest against untargeted attacks. On the CIFAR-10 dataset, **Ensemble-*n*** is even better than the **Baseline** when the confidence value of the attack is less than 20.
- Our first randomization technique **Random-Model-*n*** is generally better than the second randomization technique **Random-Weight**. However, when the attacks are untargeted, the **Random-Weight** technique demonstrates much stronger resilience to the untargeted attacks than the targeted attacks (compared to the results shown in Figure 4).
- The ensemble of **Random-Weight** DNNs is also more resilient to untargeted attacks than the single **Random-Weight** DNN. An interesting observation is that the ensemble of **Random-Weight** becomes slightly weaker against adversarial attacks when the number of random-weight DNNs increases.

4.3 L_∞ Targeted Attacks

Stochastic Activation Pruning (SAP), proposed by Dhillon et al. [15] earlier this year, introduces randomness into a neural network by randomly dropping some neurons of each layer (dropout) with a weighted probability. Carlini and Wager were able to defeat SAP by reducing its accuracy to 0% with $\epsilon = 0.031$ with their latest projected gradient descent (PGD) attack. PGD attacks measure closeness in terms of L_∞ norm. The attack takes into consideration the ineffectiveness of previous single-step attacks when facing randomness. In this new attack, they compute the gradient with the

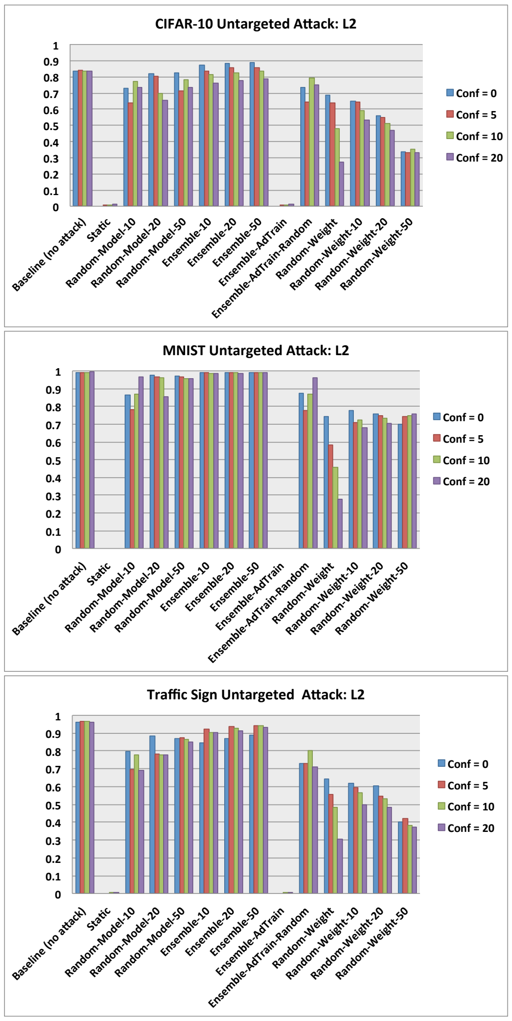

Figure 5: Results of untargeted L_2 adversarial attacks on CIFAR-10, MNIST, and Traffic Sign datasets.

expectation over instantiations of randomness. Therefore, at each iteration of gradient descent, they compute a move in the direction of

$$\sum_{i=1}^{k} \triangledown_x f(x).$$

Since larger pool sizes in our randomization techniques do not make significant differences, in this experiment, we set the pool size to 10. For our first randomization technique, smaller pool sizes also mean less computational cost. For our second randomization technique, larger pool sizes do not increase the cost, but in general reduce the resilience to attacks as the pool size grows.

4.3.1 Results on the CIFAR-10 Dataset. The severity of attacks is controlled with ϵ. Larger ϵ values give the adversary more budget to perturb the data. Figure 6 shows the impact of the ϵ value on the CIFAR-10 dataset. The original image is given on the left side, and the perturbed image is given on the right side. For different datasets, different ϵ values are needed to model the same level of attack intensity. For the CIFAR-10 dataset, we set $\epsilon = \{0.01, 0.03, 0.08\}$, and PGD search steps k to 10.

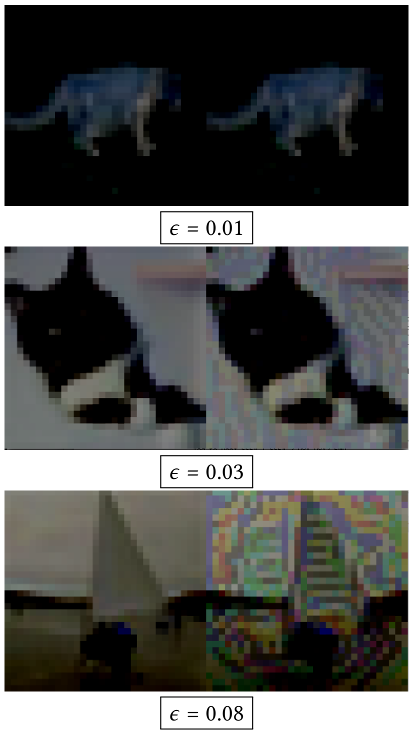

Figure 6: Quality of CIFAR-10 images after L_∞ adversarial attacks with different ϵ values.

Figure 7 shows the results of the targeted L_∞ attacks on the CIFAR-10 dataset. When $\epsilon = 0.01$ the attack is very mild. Even the accuracy of the **Static** DNN model is 74.5%, and the **Ensemble-AdTrain** technique has a slightly higher accuracy of 78.7%. The accuracy of our **Random-Model-10** (82.5%) on the perturbed samples is similar to the **Baseline** accuracy 83.7%, while the randomized **Ensemble-AdTrain-Random** has an accuracy of 83.2%. The ensemble of the DNNs in our random-model technique has a higher accuracy (87.6%) than the **Baseline** accuracy. Our random-weight technique **Random-Weight** has a slightly better accuracy (77.7%) than the **Static** model.

When attacks intensify with $\epsilon = 0.03$, the accuracy of the **Static** DNN model dropped significantly to 43.5%, while the **Ensemble-AdTrain** technique does show greater resilience with an accuracy of 54.4%. Our **Random-Model-10** demonstrates strong robustness on the perturbed samples, with an accuracy of 71.7%, while the randomized **Ensemble-AdTrain-Random** has an accuracy of 76.1%. This suggests re-training on adversarial samples does help improve the accuracy on perturbed data. The robustness of the ensemble of the DNNs in our random-model technique is very impressive, with an accuracy of 84.8%, even slightly better than the **Baseline** accuracy of 84.2%. Our random-weight technique **Random-Weight** also shows solid robustness against the targeted L_∞ attacks, with an accuracy of 77.5%, significantly better than the **Static** model.

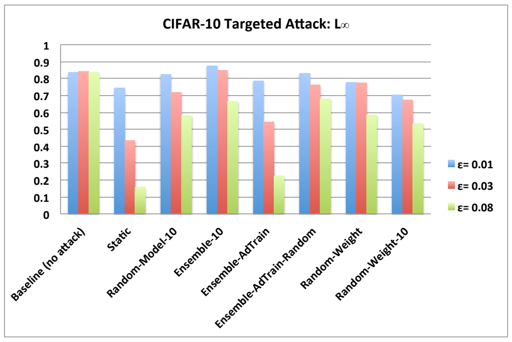

Figure 7: Results of targeted L_∞ adversarial attacks on the CIFAR-10 dataset.

The ensemble of **Random-Weight** DNNs is less impressive, but still significantly better, with an accuracy of 67.3%, than the **Static** model and the **Ensemble-AdTrain** model.

When $\epsilon = 0.08$, the attack is so severe that it is difficult for humans to recognize the objects in the images. Not surprisingly, the accuracy of the **Static** DNN model dropped to 15.8%, and similarly the accuracy of the **Ensemble-AdTrain** technique dropped to 22.5%. Our **Random-Model-10** demonstrates strong robustness to such strong attacks with an accuracy of 58.1%, while the randomized **Ensemble-AdTrain-Random** has a better accuracy of 68.1%. The ensemble of the DNNs in our random-model technique is as robust as the randomized **Ensemble-AdTrain-Random**, with an accuracy of 66.5%. Our random-weight technique **Random-Weight** also shows better robustness than the **Static** model, with an accuracy of 58.6%. The ensemble of **Random-Weight** DNNs is again more vulnerable than the single **Random-Weight** DNN, but still significantly better, with an accuracy of 53.5%, than the **Static** model and the **Ensemble-AdTrain** model.

4.3.2 Results on the MNIST Dataset. For the MNIST dataset, we set $\epsilon = \{0.2, 0.3, 0.4\}$ and PGD search steps k to 10. When $\epsilon \geq 0.5$, the entire image is filled with black pixels, and no learning models are better than random guessing. Figure 8 shows the image distortion by the targeted L_∞ attacks on the MNIST dataset with $\epsilon = \{0.2, 0.3, 0.4\}$.

Figure 9 shows the results on the MNIST dataset. When $\epsilon = 0.2$, the accuracy of the **Static** DNN model is 85.4%, about 15% lower than the **Baseline** accuracy of 99.2%. The accuracy of **Ensemble-AdTrain** is 90.9%. Again, this demonstrates the effectiveness of increasing model robustness by re-training on perturbed samples. The accuracy of our **Random-Model-10** on the perturbed samples is 95.4%, 3% lower than the accuracy (98.4%) of its ensemble counterpart. The randomized **Ensemble-AdTrain-Random** has an accuracy of 96.9%, 6% better than the plain **Ensemble-AdTrain** technique. Our random-weight technique **Random-Weight** also has a solid accuracy (96.4%), so does its ensemble counterpart (96.4%).

When $\epsilon = 0.3$, the accuracy of the **Static** DNN model dropped to 55.8%, while the accuracy of **Ensemble-AdTrain** dropped to 64.3%. Both are significant performance drop compared to the **Baseline** accuracy of 99.3%. Our **Random-Model-10** has an accuracy of 80.0%, 13.6% lower than the accuracy of its ensemble

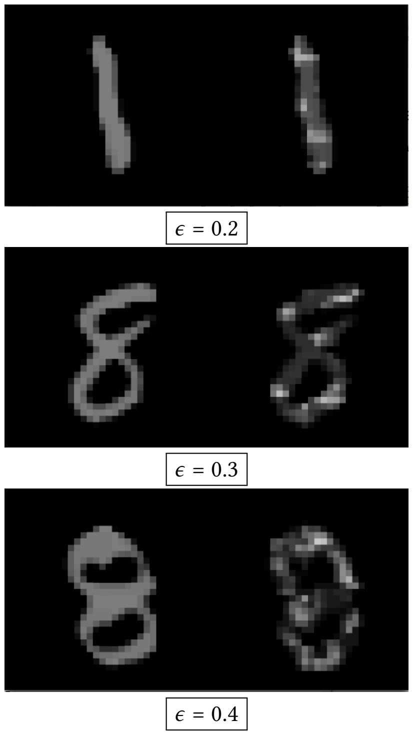

Figure 8: Quality of MNIST images after L_∞ adversarial attacks with different ϵ values.

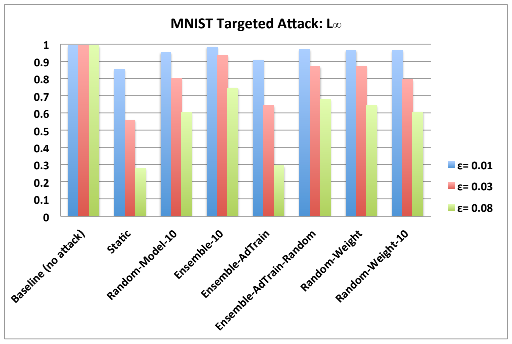

Figure 9: Results of targeted L_∞ adversarial attacks on the MNIST dataset.

counterpart **Ensemble-10**, with an accuracy of 93.6%. The reason of the large difference between the two random-model techniques is that the DNN selected for adversary happened to be the same as the one chosen for making predictions many times during the experiment with **Random-Model-10**. The randomized **Ensemble-AdTrain-Random** has an accuracy of 87.1%. Our random-weight technique **Random-Weight** has a similar accuracy of 87.3%, significantly more robust than the **Static** model. The ensemble of **Random-Weight** DNNs is still better than the **Static** model and the **Ensemble-AdTrain** model, with an accuracy of 79.4%.

When $\epsilon = 0.4$, the accuracy of the **Static** DNN model dropped to 28.0% and the accuracy of the **Ensemble-AdTrain** technique dropped to 29.4%. Our **Random-Model-10** mitigates the impact of the strong attacks with an accuracy of 60.3%, while **Ensemble-10** is again more than 14% better, with an accuracy of 74.6%, for the same reason—frequently, the model randomly chosen for prediction happened to be the model used for computing adversarial samples during the experiment with **Random-Model-10**. The randomized **Ensemble-AdTrain-Random** has a solid accuracy of 67.8%. Our random-weight technique **Random-Weight** also shows better robustness than the **Static** model, with an accuracy of 64.5%. The ensemble of ten **Random-Weight** DNNs is as usual less robust than the single **Random-Weight** DNN, with an accuracy of 60.7%, but still significantly better than the **Static** model and the **Ensemble-AdTrain** model.

4.3.3 Results on the Traffic Sign Dataset. For the German Traffic Sign dataset, we set $\epsilon = \{0.03, 0.06, 0.09\}$ to achieve similar levels of attack intensity and PGD search size $k = 10$. Figure 10 shows the distortion of the targeted L_∞ attacks on the Traffic Sign dataset with $\epsilon = \{0.03, 0.06, 0.09\}$.

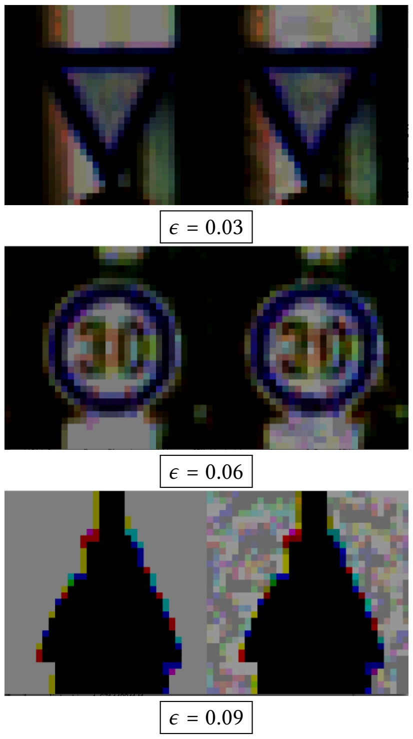

Figure 10: Quality of Traffic Sign images after L_∞ adversarial attacks with different ϵ values.

Figure 11 shows the results of the various DNN models. This is the only dataset for which our random-model technique **Random-Model-10** cannot achieve an accuracy to compete closely with the **Baseline** accuracy when the attack is targeted and relatively mild (with $\epsilon = 0.03$). The small variance (±0.025) in accuracy implies that this lagging in accuracy is not caused by randomly choosing the same DNN model as the adversary did. We will discuss the cause of this weakness by investigating the weight differential entropy between this dataset and the other two in Section 4.5.

When $\epsilon = 0.03$, the accuracy of the **Static** DNN model is 76.9%, nearly 20% lower than the **Baseline** accuracy of 96.0%. The accuracy of **Ensemble-AdTrain** is 82.0%, more than 5% better than the accuracy of the **Static** DNN model. The accuracy of our **Random-Model-10**, as mentioned earlier is 85.4%, about 10% lower than the **Baseline** accuracy, and more than 5% lower than the accuracy (91.1%) of its ensemble counterpart. The randomized **Ensemble-AdTrain-Random** has an accuracy of 90.4%, nearly 8% better than the non-randomized **Ensemble-AdTrain** technique. Our random-weight technique **Random-Weight** also has a solid accuracy of 85.6%, about 9% better than the **Static** accuracy. The ensemble of ten random-weight DNNs also has an accuracy of 85.3%.

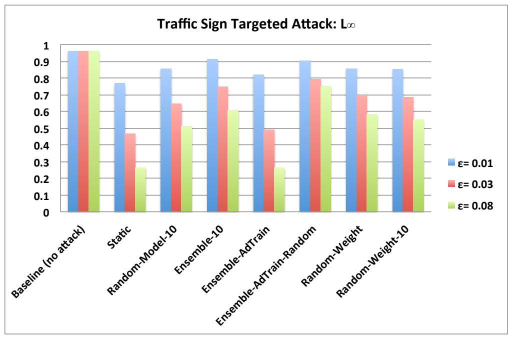

Figure 11: Results of targeted L_∞ adversarial attacks on the German Traffic Sign dataset.

When $\epsilon = 0.06$, the accuracy of the **Static** DNN model dropped to 46.5% and the accuracy of **Ensemble-AdTrain** dropped to 49.1%, approximately 50% drop from the **Baseline** accuracy of 96.0%. Our **Random-Model-10** has an accuracy of 64.7%, about 10.0% lower than the accuracy of its ensemble counterpart **Ensemble-10** at 74.9%. This can be explained by the relatively large variance in accuracy (±0.100) caused by randomly encountering the same models used by the adversary during the experiment. The randomized **Ensemble-AdTrain-Random** has an accuracy of 79.2%. Our random-weight technique **Random-Weight** has an accuracy of 69.8%, about 23% more robust than the **Static** model. The ensemble of **Random-Weight** DNNs has a similar accuracy of 68.4%.

When $\epsilon = 0.09$, both the **Static** model and the **Ensemble-AdTrain** model have a very low accuracy of about 26%. Our **Random-Model-10** has a reduced accuracy of 51.2% with a relatively large variance of ±0.102, while **Ensemble-10** is 9% more accurate at 60.7%. The randomized **Ensemble-AdTrain-Random** has a very solid accuracy of 75.0%. Our random-weight technique **Random-Weight** has better robustness than our **Random-Model-10**, with an accuracy of 58.2%. The ensemble of ten **Random-Weight** DNNs has an accuracy of 54.9%, also better than **Random-Model-10**, and significantly better than the **Static** model and the **Ensemble-AdTrain** model.

In summary, *the ensemble of the DNN models in our first randomization technique is most robust against targeted L_∞ attacks*, the single DNN random-model technique is the second robust model, followed by the single DNN random-weight technique. Ensemble adversarial training alone is as weak as the **Static** model, but combined with our first randomization technique, its robustness improved significantly and has the potential to exceed the performance of all the other techniques when the attack is severe, on condition that its large variance can be reduced by introducing more component **Ensemble-AdTrain** DNNs.

subsectionL_∞ Untargeted Attacks

We repeat the experiments presented in Section 4.3, with untargeted L_∞ attacks. For the same ϵ values, we had to set the search step to 100 to reach the same level of desired attacks. We believe the projected gradient descent technique works effectively when facing randomness, but not efficiently when facing a single static DNN model. The gradient computed as the expectation over instantiations of randomness is not optimized for attacking a single DNN model in the white-box, untargeted fashion.

Figure 12 shows the results of untargeted L_∞ attacks on the three datasets. The same conclusions can be drawn from this experiment:

- Randomization techniques are clearly more robust than the static DNN model and the *Ensemble Adversarial Training* technique, regardless of attack intensity.
- The ensemble of the DNN models in our first randomization technique is most resilient to untargeted adversarial data perturbation.
- Re-training on adversarial samples makes marginal improvement on robustness, but has a great potential to be a solid adversarial learning technique if combined with our first randomization technique.
- The spread of the distribution of the DNN models in the version space is sufficiently large such that adding a small random noise to the trained weights can significantly improve the robustness of a DNN model.

More detailed experimental results can be found in Tables 12, 13, and 14 in Appendix A.

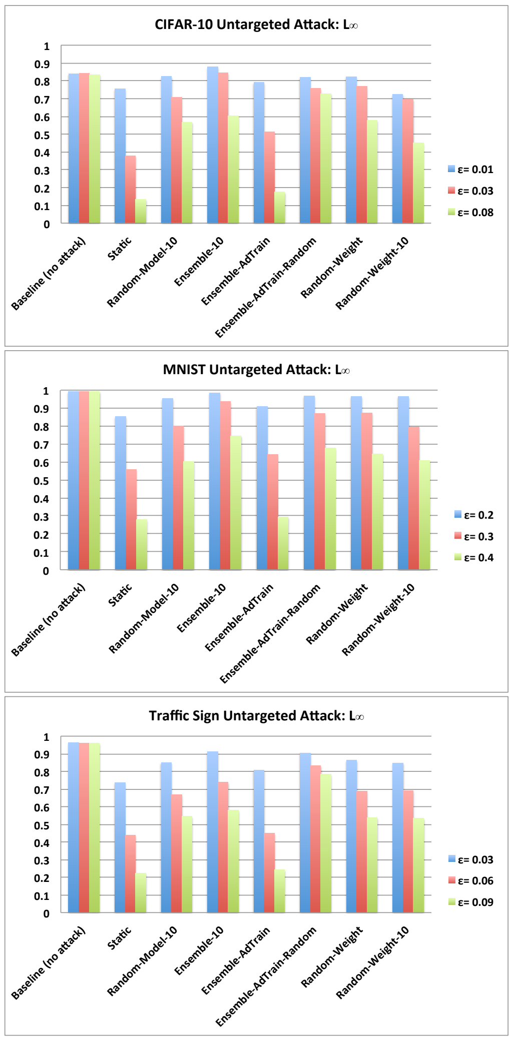

Figure 12: Results of untargeted L_∞ adversarial attacks on the CIFAR-10, MNIST and German Traffic Sign datasets.

4.4 Random-Weight DNNs on Non-perturbed Test Data

In our second randomization technique, we propose to improve the robustness of a trained DNN by adding a small Gaussian random noise to its weight. We demonstrate in our experiments that this simple but efficient technique can significantly improve the robustness of a DNN model. We now show that our **Random-Weight** DNN does not fail on the non-perturbed data, and can have the same accuracy on the test set as the original DNN model. Table 2 shows the accuracy of a trained DNN model and its randomized counterpart **Random-Weight** DNN on the non-attacked test dataset. As can be observed, adding small Gaussian random noise to the DNN model does not have any impact on its accuracy on the non-attacked test data.

Table 2: The accuracy of our Random-Weight DNNs on the non-perturbed test data.

	Original DNN	**Random-Weight DNN**
CIFAR-10	0.840 ± 0.010	0.833 ± 0.004
MNIST	0.993 ± 0.003	0.993 ± 0.001
Traffic Sign	0.963 ± 0.006	0.962 ± 0.004

4.5 Differential Entropy of DNNs for Three Datasets

We discussed in Section 3.2 how we can measure the spread of the DNN distribution in the version space by estimating the differential entropy of their weight distributions. Higher differential entropy indicates more variance in the DNN distribution, and harder for adversarial samples to transfer from one model to another. In Section 4.3.3, we observe that our random-model technique **Random-Model-10** cannot compete closely with the **Baseline** accuracy on the *German Traffic Sign* dataset, even when the attack is relatively mild. We also observe that the lagging in accuracy is not caused by choosing the same DNN model as the adversary did. Now we show that the distribution of the DNN models built on the Traffic Sign dataset has smaller differential entropy, compared to the DNN models built for the other two datasets.

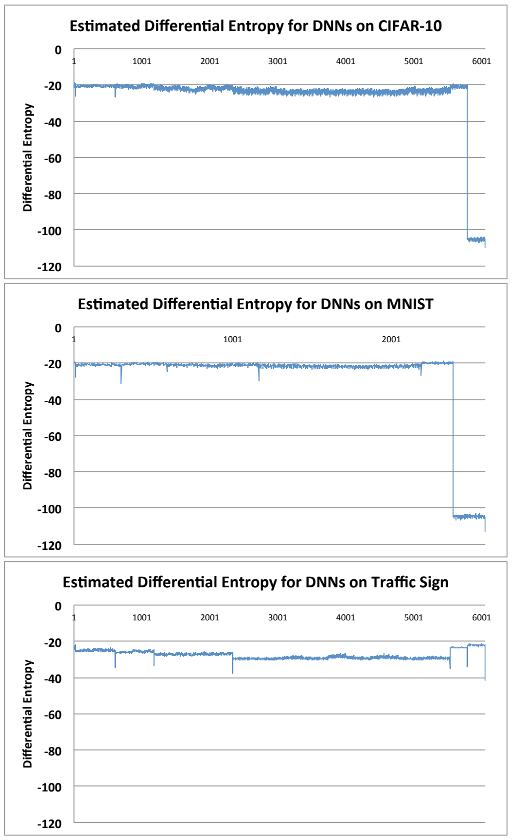

Figure 13: Differential entropy of DNN models built on the CIFAR-10, MNIST and German Traffic Sign datasets.

Figure 13 shows the estimated differential entropy for the weights to each node in the DNNs built from the three datasets. Since the true underlying distribution of the DNN models is unknown, we measure the logarithm of the absolute value of $|\Sigma|$—the determinant of the covariance matrix of the weights. As can be observed, with very few exceptions, the differential entropy of the DNNs for CIFAR and MNIST datasets is greater than that of the Traffic Sign dataset, which further supports our hypothesis that a larger variance in the DNN distribution implies more difficult attack through transferability. It also explains the question we raised in Section 4.3.3 that on the *Traffic Sign* dataset our first randomization technique cannot closely approach the **Baseline** accuracy, unlike on the other two datasets when the attack is relatively mild.

5 DISCUSSIONS

Naturally, a subsequent question to ask is how to break the randomization techniques from the adversary's perspective. The adversary can query our randomized DNNs and get a set of data labeled. One thing the adversary can do with the dataset is to build a model on that dataset and compute adversarial samples from the learned model. The other thing the adversary can do is to train an ensemble of n models with that dataset and compute adversarial samples from the ensemble model.

We argue that none of these strategies would be effective, as long as we can get a similar set of data as the adversary did. This is easier for us since we have all the models and we can simulate adversary's probing behavior to get a similar set of data. Suppose the adversary builds a new model f on dataset D, we can build a set of different models on a similar dataset $D' \approx D$ with stochastic gradient descent. Even if the adversary builds an ensemble, we can build a pool of different ensembles from that dataset. Now the problem circles back to the same one we tackle in this paper: the adversary has perfect knowledge of one model (single or ensemble), and we keep a set of various models trained on the same/similar data from the adversary. As long as variances in decision boundaries are large enough, our randomization techniques would be robust to adversarial attacks. There are two additional methods we can use to further improve the robustness of our randomization techniques: 1.) we can combine our two randomization techniques: at each query request, we randomly select a model and add a small random noise to its weights before we answer the query; and 2.) we can re-train a set of DNNs on the adversarial samples we compute and use our randomization techniques with the re-trained DNNs. We have demonstrated in our experiments that the latter approach has a great potential to exceed all the other randomization techniques we have presented earlier.

Of course this cat and mouse game cannot go on forever. Eventually, the adversary would stop further attacking to preserve adversarial utility. If the attack budget is unlimited, there is no defense that could effectively mitigate such arbitrary attacks.

We experiment on the ideas of attacking the ensembles of DNNs, and defending them by adding random noise to the weights of the DNNs as we discussed earlier. Successfully attacking the ensemble evidently requires much greater data perturbation than attacking a single DNN, as shown in Figure 14 in the case of targeted L_2 attacks and Figure 15 in the case of untargeted L_2 attacks. Given a confidence value, the blue bar shows the perturbation required to defeat a single DNN, while the red bar shows the required perturbation for defeating an ensemble of 10 DNNs.

We create an ensemble of DNNs by randomly selecting 10 DNNs from a pool of 100 DNNs. When facing significantly greater data perturbation by targeted attacks, we observe that without randomization, that is, the ensemble for attack E_a is the same as the ensemble for prediction E_p, any ensemble of DNNs ($E_p = E_a$) would be foiled completely, with its accuracy disappearing in the plots under the label of **static_ensemble** in Figure 16. The last bar in each group shows the accuracy on the unattacked images. If we predict with a different ensemble (i.e. $E_p \neq E_a$) by randomly selecting another 10 DNNs (with replacement) from the pool, when the attack confidence is zero ($Conf = 0$) we can successfully mitigate the attacks, labeled as **random_ensemble** in the figure. This type of randomization becomes less effective when the attack confidence is larger, especially when $Conf \geq 10$. This is understandable since the distortion on the original image is noticeably intensified.

We also tested the idea of predicting with one of the DNNs in E_p after E_a has been attacked. The results are labeled as **random_single** in Figure 16. When $Conf = 0$ a single DNN is less resilient to attacks compared to an ensemble of DNN. However, when $Conf \geq 5$ the majority voting scheme breaks down because

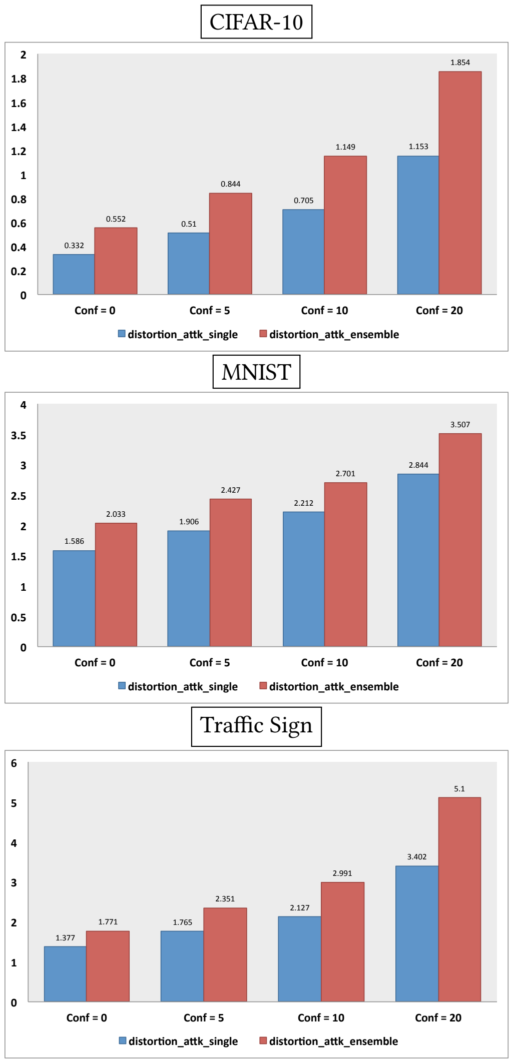

Figure 14: Data perturbation required for successful targeted L_2 attacks against ensemble DNNs on CIFAR-10, MNIST, and Traffic Sign datasets.

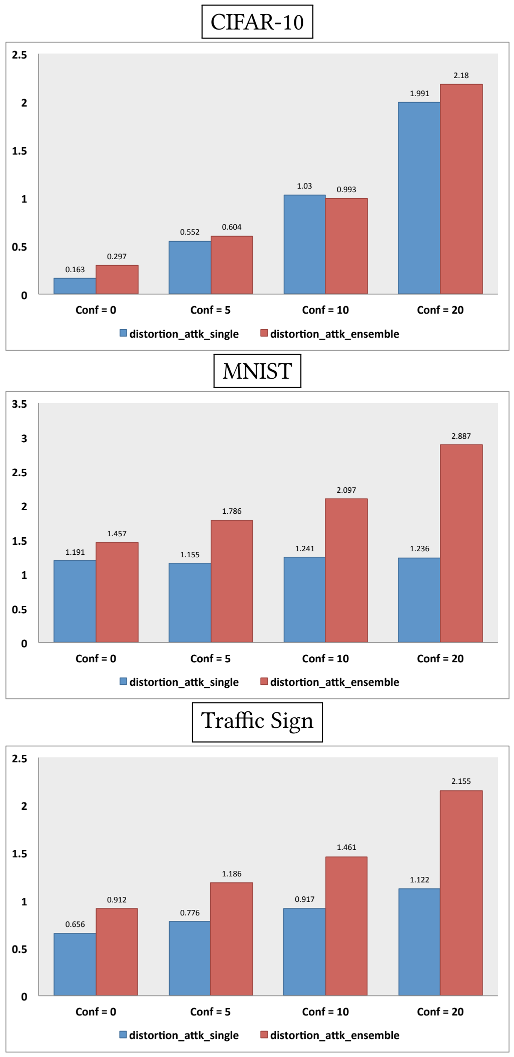

Figure 15: Data perturbation required for successful untargeted L_2 attacks against ensemble DNNs on CIFAR-10, MNIST, and Traffic Sign datasets.

of the weak component DNNs, and the ensemble becomes worse than the single DNN.

We further introduce randomization by adding noise to the weights of each of the DNNs in the ensemble, shown as **random_std**_x in Figure 16 where x is the value of the standard deviation used to generate the random noise. As can be observed, when x is small, the accuracy on the unattacked data is high and the randomization scheme works well when $Conf = 0$. As x increases, the accuracy on the unattacked data drops, so is the accuracy of the randomized DNN ensemble for $Conf = 0$. On the other hand, the accuracy of the randomized DNN ensemble increases for $Conf > 0$. Therefore, it is important to choose an x to keep a good balance between accuracy on the attacked and the unattacked data.

Among the three datasets, our randomization with random weights technique works extremely well on the MNIST dataset, followed by the CIFAR-10 dataset, and Traffic Sign dataset. For the Traffic Sign dataset, we already know its differential entropy is much lower, therefore randomization is least effective on this dataset although it is still much more resilient than **static_ensemble** and **random_ensemble**.

Similar results are obtained for the untargeted attacks, as shown in Figure 17.

6 CONCLUSIONS

We investigate the nature of transferability of perturbed data. We believe that transferability should not be automatically assumed in a learning task. It strongly ties to the spread of the DNN model distribution in the version space and the severity of adversarial attacks. We present two randomization techniques for robust DNN learning. Our first randomization technique involves training a

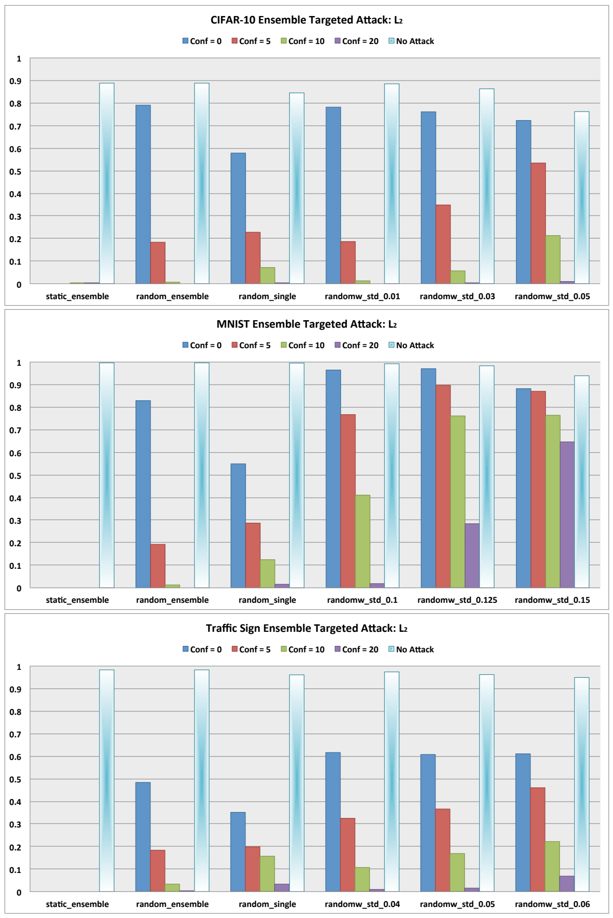

Figure 16: Accuracy of different DNN classifiers facing targeted L_2 attacks on CIFAR-10, MNIST, and Traffic Sign datasets.

pool of DNNs. For each query request, a DNN model randomly selected from the pool is used to answer the query. We assume the adversary has the complete knowledge of one of the DNN models, and the rest are kept secret to the adversary. We also investigate the robustness of the ensemble of the subset of the DNN models. In our second randomization technique, we simply train one DNN model, and add a small Gaussian random noise to the weights of the model. This approach is computationally efficient. Our experimental results demonstrate that the ensemble of the DNNs in our first randomization technique is most robust, sometimes exceeding the accuracy of the DNN model on non-attacked samples. Our second randomization technique is also significantly more robust against adversarial attacks, including both targeted and untargeted attacks. We demonstrate that re-training on the adversarial samples alone will not improve robustness, but combined with our first randomization technique, the ensemble adversarial training may have a great potential of being the most robust technique.

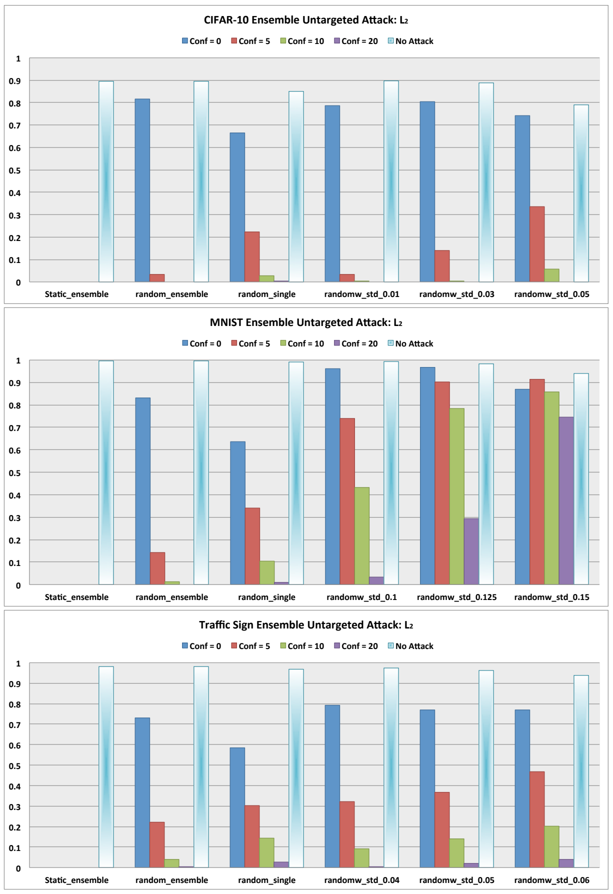

Figure 17: Accuracy of different DNN classifiers facing untargeted L_2 attacks on CIFAR-10, MNIST, and Traffic Sign datasets.

REFERENCES

[1] Anish Athalye, Nicholas Carlini, and David A. Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. *CoRR* abs/1802.00420 (2018).

[2] Peter Auer and Nicolò Cesa-Bianchi. 1998. On-line learning with malicious noise and the closure algorithm. *Annals of Mathematics and Artificial Intelligence* 23, 1-2 (Jan. 1998), 83–99.

[3] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. 2010. The Security of Machine Learning. *Mach. Learn.* 81, 2 (Nov. 2010), 121–148.

[4] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. 2016. Measuring Neural Net Robustness with Constraints. In *NIPS*. 2613–2621.

[5] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. 2016. Dimensionality Reduction as a Defense against Evasion Attacks on Machine Learning Classifiers. *CoRR* abs/1704.02654 (2016).

[6] M. Bruckner and T. Scheffer. 2011. Stackelberg Games for Adversarial Prediction Problems. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

[7] Nader H. Bshouty, Nadav Eiron, and Eyal Kushilevitz. 1999. PAC learning with nasty noise. *Theoretical Computer Science* 288 (1999), 2002.

[8] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. 2018. Thermometer Encoding: One Hot Way To Resist Adversarial Examples. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S18Su--CW

[9] Nicholas Carlini and David Wagner. 2017. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec '17)*. ACM, New York, NY, USA, 3–14. https://doi.org/10.1145/3128572.3140444

[10] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. *2017 IEEE Symposium on Security and Privacy (SP)* (2017), 39–57.

[11] Krzysztof Chalupka, Pietro Perona, and Frederick Eberhardt. 2015. Visual Causal Feature Learning. In *UAI*. AUAI Press, 181–190.

[12] Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA.

[13] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. 2004. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '04)*. ACM, New York, NY, USA, 99–108.

[14] O. Dekel and O. Shamir. 2008. Learning to classify with missing and corrupted features. In *Proceedings of the International Conference on Machine Learning*. ACM, 216–223.

[15] Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. 2018. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1uR4GZRZ

[16] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner. 2017. Detecting Adversarial Samples from Artifacts. *ArXiv e-prints* (March 2017). arXiv:stat.ML/1703.00410

[17] Amir Globerson and Sam Roweis. 2006. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd international conference on Machine learning (ICML '06)*. ACM, 353–360.

[18] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*. http://arxiv.org/abs/1412.6572

[19] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick D. McDaniel. 2017. On the (Statistical) Detection of Adversarial Examples. *CoRR* abs/1702.06280 (2017).

[20] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. 2018. Countering Adversarial Images using Input Transformations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SyJ7ClWCb

[21] Dan Hendrycks and Kevin Gimpel. 2017. Early Methods for Detecting Adversarial Images. In *International Conference on Learning Representations Workshop*. https://arxiv.org/abs/1608.00530

[22] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. 2013. Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*.

[23] Murat Kantarcioglu, Bowei Xi, and Chris Clifton. 2011. Classifier evaluation and attribute selection against active adversaries. *Data Min. Knowl. Discov.* 22 (January 2011), 291–335. Issue 1-2.

[24] Michael Kearns and Ming Li. 1993. Learning in the Presence of Malicious Errors. *SIAM J. Comput.* 22 (1993), 807–837.

[25] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2009. CIFAR-10 (Canadian Institute for Advanced Research). (2009). http://www.cs.toronto.edu/~kriz/cifar.html

[26] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

[27] Xin Li and Fuxin Li. 2016. Adversarial Examples Detection in Deep Networks with Convolutional Filter Statistics. *CoRR* abs/1612.07767 (2016).

[28] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2017. Delving in to transferable adversarial examples and black-box attacks. In *International Conference on Learning Representations*. https://arxiv.org/abs/1611.02770

[29] Daniel Lowd and Christopher Meek. 2005. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining (KDD '05)*. 641–647.

[30] Xingjun Ma, Bo Li, Yisen Wang, Sarah M. Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Michael E. Houle, Dawn Song, and James Bailey. 2018. Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality. In *International Conference on Learning Representations*. https://openreview.net/forum?id=B1gJ1L2aW

[31] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJzIBfZAb

[32] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277* (2016).

[33] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks Against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17)*. ACM, New York, NY, USA, 506–519. https://doi.org/10.1145/3052973.3053009

[34] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. instillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy (SP)*.

[35] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 582–597.

[36] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. 2018. Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BkJ3ibb0-

[37] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. 2018. PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJUYGxbCW

[38] Jörg Stoye. 2011. Statistical Decisions Under Ambiguity. *Theory and Decision* 70, 2 (2011), 129–148.

[39] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*. http://arxiv.org/abs/1312.6199

[40] Florian Tramer, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2018. Ensemble Adversarial Training: Attacks AND Defenses. In *International Conference on Learning Representations*. https://arxiv.org/pdf/1704.03453.pdf

[41] Abraham Wald. 1939. Contributions to the Theory of Statistical Estimation and Testing Hypotheses. *Ann. Math. Statist.* 10, 4 (12 1939), 299–326. https://doi.org/10.1214/aoms/1177732144

[42] Abraham Wald. 1945. Statistical Decision Functions Which Minimize the Maximum Risk. *Annals of Mathematics* 46 (1945), 265–280.

[43] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. 2018. Mitigating Adversarial Effects Through Randomization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Sk9yuql0Z

[44] Stephan Zheng, Yang Song, Thomas Leung, and Ian J. Goodfellow. 2016. Improving the Robustness of Deep Neural Networks via Stability Training. In *CVPR*. IEEE Computer Society, 4480–4488.

[45] Yan Zhou, Murat Kantarcioglu, Bhavani Thuraisingham, and Bowei Xi. 2012. Adversarial support vector machine learning. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '12)*. ACM, New York, NY, USA, 1059–1067. https://doi.org/10.1145/2339530.2339697

A TABLES OF DETAILED EXPERIMENTAL RESULTS

We present all the tables that contain the detailed experimental results in this section. For L_2 attacks with confidence values of 0, 5, 10, and 20, each table lists the **Baseline** accuracy on the original data, the **Static** accuracy on perturbed data when the same DNNs are used for both attack probing and prediction, the accuracy of our first randomization scheme **Random-Model-*n*** where $n = 10, 20, 50$, the accuracy of the ensemble of the pool of DNNs in our first randomization scheme **Ensemble-*n*** where $n = 10, 20, 50$, the accuracy of the ensemble adversarial training technique **Ensemble-AdTrain** and **Ensemble-AdTrain** combined with our first randomization scheme **Ensemble-AdTrain-Random**, and the accuracy of our second randomization scheme and its ensembles **Random-Weight** and **Random-Weight-*n*** where $n = 10, 20, 50$. For L_∞ attacks with various ϵ values, each tables lists the same results except for the pool size of 20 and 50 in our randomization techniques.

ACKNOWLEDGMENTS

The research reported herein was supported in part by NIH award 1R01HG006844, NSF awards CNS-1111529, CICI-1547324, and IIS-1633331 and ARO award W911NF-17- 1-0356.

Table 3: Cifar-10 Targeted L2 Attack Results

Algorithm	Conf = 0	Conf = 5	Conf = 10	Conf = 20
Baseline (no attack)	0.846 ± 0.009	0.846 ± 0.007	0.840 ± 0.013	0.839 ± 0.012
Static	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.001 ± 0.001
Random-Model-10	0.816 ± 0.010	0.789 ± 0.010	0.729 ± 0.020	0.589 ± 0.017
Random-Model-20	0.810 ± 0.011	0.708 ± 0.236	0.734 ± 0.013	0.464 ± 0.233
Random-Model-50	0.807 ± 0.015	0.763 ± 0.013	0.727 ± 0.013	0.575 ± 0.018
Ensemble-10	0.873 ± 0.008	0.845 ± 0.009	0.792 ± 0.007	0.637 ± 0.025
Ensemble-20	0.883 ± 0.011	0.856 ± 0.012	0.815 ± 0.009	0.618 ± 0.021
Ensemble-50	0.891 ± 0.009	0.869 ± 0.010	0.817 ± 0.015	0.630 ± 0.017
Ensemble-AdTrain	0.000 ± 0.000	0.000 ± 0.000	0.001 ± 0.001	0.002 ± 0.002
Ensemble-AdTrain-Random	0.801 ± 0.012	0.786 ± 0.011	0.752 ± 0.018	0.640 ± 0.050
Random-Weight	0.697 ± 0.025	0.089 ± 0.021	0.002 ± 0.002	0.071 ± 0.003
Random-Weight-10	0.624 ± 0.021	0.504 ± 0.033	0.306 ± 0.047	0.543 ± 0.018
Random-Weight-20	0.607 ± 0.028	0.485 ± 0.031	0.550 ± 0.037	0.193 ± 0.037
Random-Weight-50	0.614 ± 0.055	0.500 ± 0.037	0.342 ± 0.034	0.289 ± 0.070
Mean Distortion	0.332 ± 0.005	0.510 ± 0.010	0.705 ± 0.020	1.153 ± 0.041
Ensemble-AdTrain Distortion	0.420 ± 0.016	0.671 ± 0.033	0.948 ± 0.070	1.611 ± 0.157

Table 4: MNIST Targeted L2 Attack Results

Algorithm	Conf = 0	Conf = 5	Conf = 10	Conf = 20
Baseline (no attack)	0.993 ± 0.002	0.992 ± 0.003	0.993 ± 0.001	0.990 ± 0.003
Static	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Random-Model-10	0.964 ± 0.016	0.924 ± 0.025	0.851 ± 0.069	0.529 ± 0.207
Random-Model-20	0.845 ± 0.283	0.906 ± 0.032	0.827 ± 0.065	0.590 ± 0.200
Random-Model-50	0.952 ± 0.014	0.925 ± 0.018	0.825 ± 0.055	0.596 ± 0.085
Ensemble-10	0.991 ± 0.002	0.972 ± 0.009	0.922 ± 0.022	0.633 ± 0.057
Ensemble-20	0.992 ± 0.002	0.971 ± 0.009	0.916 ± 0.037	0.702 ± 0.053
Ensemble-50	0.991 ± 0.004	0.978 ± 0.004	0.929 ± 0.018	0.674 ± 0.073
Ensemble-AdTrain	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Ensemble-AdTrain-Random	0.954 ± 0.025	0.944 ± 0.034	0.911 ± 0.060	0.801 ± 0.269
Random-Weight	0.663 ± 0.096	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Random-Weight-10	0.733 ± 0.104	0.316 ± 0.147	0.051 ± 0.043	0.000 ± 0.000
Random-Weight-20	0.809 ± 0.039	0.482 ± 0.104	0.177 ± 0.084	0.014 ± 0.014
Random-Weight-50	0.696 ± 0.110	0.580 ± 0.119	0.384 ± 0.111	0.117 ± 0.068
Mean Distortion	1.586 ± 0.052	1.906 ± 0.049	2.212 ± 0.060	2.844 ± 0.093
Ensemble-AdTrain Distortion	1.473 ± 0.219	1.697 ± 0.257	1.965 ± 0.272	2.190 ± 0.208

Table 5: Traffic Sign Targeted L2 Attack Results

Algorithm	Conf = 0	Conf = 5	Conf = 10	Conf = 20
Baseline (no attack)	0.961 ± 0.006	0.960 ± 0.006	0.963 ± 0.006	0.958 ± 0.003
Static	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.001 ± 0.001
Random-Model-10	0.735 ± 0.029	0.528 ± 0.266	0.465 ± 0.235	0.417 ± 0.038
Random-Model-20	0.662 ± 0.223	0.610 ± 0.209	0.421 ± 0.278	0.332 ± 0.129
Random-Model-50	0.740 ± 0.042	0.687 ± 0.037	0.577 ± 0.067	0.334 ± 0.168
Ensemble-10	0.848 ± 0.015	0.778 ± 0.023	0.681 ± 0.027	0.439 ± 0.051
Ensemble-20	0.871 ± 0.019	0.809 ± 0.025	0.722 ± 0.025	0.485 ± 0.043
Ensemble-50	0.887 ± 0.019	0.822 ± 0.026	0.717 ± 0.052	0.479 ± 0.052
Ensemble-AdTrain	0.000 ± 0.000	0.003 ± 0.002	0.003 ± 0.002	0.008 ± 0.003
Ensemble-AdTrain-Random	0.897 ± 0.017	0.677 ± 0.338	0.639 ± 0.319	0.671 ± 0.023
Random-Weight	0.414 ± 0.044	0.012 ± 0.008	0.001 ± 0.001	0.001 ± 0.002
Random-Weight-10	0.432 ± 0.038	0.255 ± 0.031	0.097 ± 0.027	0.009 ± 0.009
Random-Weight-20	0.405 ± 0.051	0.289 ± 0.034	0.202 ± 0.026	0.042 ± 0.015
Random-Weight-50	0.443 ± 0.042	0.226 ± 0.035	0.182 ± 0.039	0.089 ± 0.022
Mean Distortion	1.377 ± 0.066	1.765 ± 0.127	2.127 ± 0.121	3.402 ± 0.465
Ensemble-AdTrain Distortion	1.200 ± 0.040	2.050 ± 0.153	2.715 ± 0.163	4.263 ± 0.304

Table 6: Cifar-10 Untargeted L2 Attack Results

Algorithm	Conf = 0	Conf = 5	Conf = 10	Conf = 20
Baseline (no attack)	0.835 ± 0.011	0.840 ± 0.010	0.838 ± 0.019	0.838 ± 0.014
Static	0.000 ± 0.000	0.003 ± 0.002	0.006 ± 0.002	0.011 ± 0.006
Random-Model-10	0.729 ± 0.243	0.639 ± 0.317	0.773 ± 0.017	0.733 ± 0.014
Random-Model-20	0.822 ± 0.017	0.803 ± 0.014	0.697 ± 0.232	0.653 ± 0.209
Random-Model-50	0.825 ± 0.010	0.713 ± 0.237	0.780 ± 0.009	0.734 ± 0.016
Ensemble-10	0.873 ± 0.008	0.835 ± 0.014	0.815 ± 0.011	0.762 ± 0.017
Ensemble-20	0.883 ± 0.011	0.858 ± 0.011	0.824 ± 0.012	0.775 ± 0.012
Ensemble-50	0.891 ± 0.009	0.858 ± 0.007	0.836 ± 0.006	0.786 ± 0.015
Ensemble-AdTrain	0.000 ± 0.000	0.003 ± 0.001	0.004 ± 0.002	0.014 ± 0.004
Ensemble-AdTrain-Random	0.736 ± 0.246	0.643 ± 0.320	0.791 ± 0.014	0.751 ± 0.016
Random-Weight	0.686 ± 0.016	0.639 ± 0.020	0.481 ± 0.024	0.274 ± 0.031
Random-Weight-10	0.651 ± 0.029	0.645 ± 0.021	0.589 ± 0.031	0.531 ± 0.037
Random-Weight-20	0.558 ± 0.046	0.548 ± 0.053	0.513 ± 0.052	0.471 ± 0.049
Random-Weight-50	0.335 ± 0.052	0.331 ± 0.020	0.354 ± 0.070	0.331 ± 0.064
Mean Distortion	0.163 ± 0.006	0.552 ± 0.068	1.030 ± 0.083	1.991 ± 0.122
Ensemble-AdTrain Distortion	0.162 ± 0.005	0.582 ± 0.063	0.954 ± 0.067	1.854 ± 0.155

Table 7: MNIST Untargeted L2 Attack Results

Algorithm	Conf = 0	Conf = 5	Conf = 10	Conf = 20
Baseline (no attack)	0.992 ± 0.003	0.992 ± 0.002	0.992 ± 0.004	0.993 ± 0.002
Static	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Random-Model-10	0.863 ± 0.289	0.781 ± 0.391	0.868 ± 0.290	0.967 ± 0.009
Random-Model-20	0.977 ± 0.006	0.965 ± 0.014	0.959 ± 0.018	0.854 ± 0.285
Random-Model-50	0.972 ± 0.009	0.968 ± 0.010	0.955 ± 0.011	0.958 ± 0.014
Ensemble-10	0.991 ± 0.002	0.988 ± 0.003	0.984 ± 0.004	0.985 ± 0.005
Ensemble-20	0.992 ± 0.002	0.989 ± 0.003	0.988 ± 0.004	0.987 ± 0.006
Ensemble-50	0.991 ± 0.004	0.991 ± 0.003	0.989 ± 0.003	0.988 ± 0.004
Ensemble-AdTrain	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Ensemble-AdTrain-Random	0.874 ± 0.291	0.779 ± 0.389	0.868 ± 0.290	0.963 ± 0.009
Random-Weight	0.742 ± 0.071	0.581 ± 0.090	0.456 ± 0.114	0.279 ± 0.142
Random-Weight-10	0.779 ± 0.078	0.711 ± 0.111	0.723 ± 0.097	0.681 ± 0.051
Random-Weight-20	0.756 ± 0.100	0.749 ± 0.091	0.731 ± 0.079	0.703 ± 0.083
Random-Weight-50	0.699 ± 0.080	0.742 ± 0.071	0.748 ± 0.070	0.759 ± 0.093
Mean Distortion	1.191 ± 0.059	1.155 ± 0.067	1.241 ± 0.056	1.236 ± 0.051
Ensemble-AdTrain Distortion	1.433 ± 0.047	1.445 ± 0.031	1.479 ± 0.040	1.540 ± 0.041

Table 8: Traffic Sign Untargeted L2 Attack Results

Algorithm	Conf = 0	Conf = 5	Conf = 10	Conf = 20
Baseline (no attack)	0.962 ± 0.007	0.964 ± 0.004	0.967 ± 0.005	0.962 ± 0.005
Static	0.000 ± 0.000	0.000 ± 0.001	0.001 ± 0.001	0.001 ± 0.001
Random-Model-10	0.795 ± 0.265	0.695 ± 0.348	0.779 ± 0.261	0.692 ± 0.346
Random-Model-20	0.886 ± 0.042	0.781 ± 0.261	0.778 ± 0.260	0.780 ± 0.260
Random-Model-50	0.871 ± 0.029	0.875 ± 0.025	0.865 ± 0.025	0.850 ± 0.033
Ensemble-10	0.848 ± 0.015	0.925 ± 0.014	0.903 ± 0.020	0.905 ± 0.020
Ensemble-20	0.871 ± 0.019	0.938 ± 0.012	0.930 ± 0.006	0.913 ± 0.020
Ensemble-50	0.887 ± 0.019	0.944 ± 0.009	0.940 ± 0.009	0.934 ± 0.011
Ensemble-AdTrain	0.000 ± 0.000	0.000 ± 0.001	0.001 ± 0.001	0.001 ± 0.002
Ensemble-AdTrain-Random	0.731 ± 0.366	0.729 ± 0.365	0.804 ± 0.269	0.710 ± 0.355
Random-Weight	0.642 ± 0.028	0.557 ± 0.051	0.482 ± 0.041	0.305 ± 0.034
Random-Weight-10	0.618 ± 0.055	0.594 ± 0.047	0.566 ± 0.040	0.497 ± 0.037
Random-Weight-20	0.603 ± 0.031	0.548 ± 0.040	0.532 ± 0.030	0.483 ± 0.036
Random-Weight-50	0.402 ± 0.043	0.419 ± 0.042	0.381 ± 0.074	0.373 ± 0.063
Mean Distortion	0.656 ± 0.017	0.776 ± 0.044	0.917 ± 0.063	1.122 ± 0.108
Ensemble-AdTrain Distortion	0.600 ± 0.026	0.722 ± 0.032	0.827 ± 0.075	1.133 ± 0.044

Table 9: CIFAR-10 Targeted L_∞ Attack Results

Algorithm	ϵ= **0.01**	ϵ= **0.03**	ϵ= **0.08**
Baseline (no attack)	0.837 ± 0.009	0.842 ± 0.013	0.837 ± 0.015
Static	0.745 ± 0.020	0.435 ± 0.021	0.158 ± 0.015
Random-Model-10	0.825 ± 0.010	0.717 ± 0.137	0.581 ± 0.144
Ensemble-10	0.876 ± 0.010	0.848 ± 0.012	0.665 ± 0.018
Ensemble-AdTrain	0.787 ± 0.008	0.544 ± 0.017	0.225 ± 0.017
Ensemble-AdTrain-Random	0.832 ± 0.010	0.761 ± 0.118	0.681 ± 0.153
Random-Weight	0.777 ± 0.018	0.775 ± 0.019	0.586 ± 0.025
Random-Weight-10	0.704 ± 0.017	0.673 ± 0.024	0.535 ± 0.021
Mean Distortion	0.544 ± 0.000	1.596 ± 0.001	4.063 ± 0.003
Ensemble-AdTrain Distortion	0.545 ± 0.000	1.601 ± 0.001	4.066 ± 0.004

Table 10: MNIST Targeted L_∞ Attack Results

Algorithm	ϵ= **0.2**	ϵ= **0.3**	ϵ= **0.4**
Baseline (no attack)	0.992 ± 0.003	0.993 ± 0.003	0.992 ± 0.002
Static	0.854 ± 0.022	0.558 ± 0.025	0.280 ± 0.021
Random-Model-10	0.954 ± 0.039	0.800 ± 0.140	0.603 ± 0.118
Ensemble-10	0.984 ± 0.006	0.936 ± 0.014	0.746 ± 0.052
Ensemble-AdTrain	0.909 ± 0.017	0.643 ± 0.030	0.294 ± 0.064
Ensemble-AdTrain-Random	0.969 ± 0.020	0.871 ± 0.105	0.678 ± 0.175
Random-Weight	0.964 ± 0.012	0.873 ± 0.044	0.645 ± 0.071
Random-Weight-10	0.964 ± 0.013	0.794 ± 0.128	0.607 ± 0.085
Mean Distortion	5.504 ± 0.002	8.174 ± 0.004	10.784 ± 0.004
Ensemble-AdTrain Distortion	5.508 ± 0.002	8.181 ± 0.004	10.784 ± 0.005

Table 11: Traffic Sign Targeted L_∞ Attack Results

Algorithm	ϵ= **0.03**	ϵ= **0.06**	ϵ= **0.09**
Baseline (no attack)	0.960 ± 0.005	0.961 ± 0.005	0.961 ± 0.008
Static	0.769 ± 0.020	0.465 ± 0.019	0.264 ± 0.016
Random-Model-10	0.854 ± 0.025	0.647 ± 0.100	0.512 ± 0.102
Ensemble-10	0.911 ± 0.014	0.749 ± 0.020	0.607 ± 0.023
Ensemble-AdTrain	0.820 ± 0.018	0.491 ± 0.025	0.263 ± 0.023
Ensemble-AdTrain-Random	0.904 ± 0.023	0.792 ± 0.158	0.750 ± 0.171
Random-Weight	0.856 ± 0.032	0.698 ± 0.047	0.582 ± 0.035
Random-Weight-10	0.853 ± 0.019	0.684 ± 0.019	0.549 ± 0.033
Mean Distortion	1.624 ± 0.001	3.167 ± 0.003	4.642 ± 0.007
Ensemble-AdTrain Distortion	1.620 ± 0.001	3.158 ± 0.001	4.626 ± 0.003

Table 12: CIFAR-10 Untargeted L_∞ Attack Results

Algorithm	ϵ= **0.01**	ϵ= **0.03**	ϵ= **0.08**
Baseline (no attack)	0.840 ± 0.010	0.841 ± 0.009	0.835 ± 0.014
Static	0.754 ± 0.021	0.378 ± 0.025	0.135 ± 0.009
Random-Model-10	0.825 ± 0.017	0.706 ± 0.170	0.568 ± 0.041
Ensemble-10	0.879 ± 0.009	0.844 ± 0.014	0.602 ± 0.023
Ensemble-AdTrain	0.791 ± 0.017	0.513 ± 0.021	0.176 ± 0.013
Ensemble-AdTrain-Random	0.820 ± 0.018	0.758 ± 0.129	0.728 ± 0.020
Random-Weight	0.823 ± 0.011	0.769 ± 0.022	0.579 ± 0.032
Random-Weight-10	0.724 ± 0.035	0.695 ± 0.029	0.451 ± 0.027
Mean Distortion	0.545 ± 0.000	1.606 ± 0.001	4.111 ± 0.007
Ensemble-AdTrain Distortion	0.545 ± 0.000	1.608 ± 0.001	4.108 ± 0.004

Table 13: MNIST Untargeted L_∞ Attack Results

Algorithm	ϵ= **0.2**	ϵ= **0.3**	ϵ= **0.4**
Baseline (no attack)	0.992 ± 0.003	0.992 ± 0.003	0.992 ± 0.002
Static	0.801 ± 0.027	0.363 ± 0.025	0.133 ± 0.009
Random-Model-10	0.962 ± 0.009	0.841 ± 0.020	0.554 ± 0.146
Ensemble-10	0.981 ± 0.004	0.912 ± 0.009	0.635 ± 0.032
Ensemble-AdTrain	0.860 ± 0.016	0.479 ± 0.050	0.146 ± 0.023
Ensemble-AdTrain-Random	0.971 ± 0.007	0.889 ± 0.043	0.573 ± 0.157
Random-Weight	0.960 ± 0.015	0.848 ± 0.042	0.568 ± 0.068
Random-Weight-10	0.956 ± 0.014	0.780 ± 0.122	0.526 ± 0.073
Mean Distortion	5.521 ± 0.001	8.220 ± 0.003	10.884 ± 0.004
Ensemble-AdTrain Distortion	5.523 ± 0.002	8.227 ± 0.002	10.892 ± 0.006

Table 14: Traffic Sign Untargeted L_∞ Attack Results

Algorithm	ϵ= **0.03**	ϵ= **0.06**	ϵ= **0.09**
Baseline (no attack)	0.963 ± 0.006	0.960 ± 0.007	0.961 ± 0.007
Static	0.737 ± 0.028	0.440 ± 0.025	0.224 ± 0.030
Random-Model-10	0.850 ± 0.044	0.669 ± 0.073	0.546 ± 0.032
Ensemble-10	0.913 ± 0.012	0.739 ± 0.023	0.580 ± 0.026
Ensemble-AdTrain	0.808 ± 0.008	0.450 ± 0.021	0.242 ± 0.027
Ensemble-AdTrain-Random	0.904 ± 0.047	0.832 ± 0.132	0.784 ± 0.023
Random-Weight	0.864 ± 0.010	0.689 ± 0.024	0.538 ± 0.036
Random-Weight-10	0.848 ± 0.027	0.691 ± 0.042	0.534 ± 0.015
Mean Distortion	1.625 ± 0.001	3.197 ± 0.004	4.720 ± 0.004
Ensemble-AdTrain Distortion	1.626 ± 0.000	3.191 ± 0.001	4.699 ± 0.002