2017 IEEE Symposium on Security and Privacy

Towards Evaluating the Robustness of Neural Networks

Nicholas Carlini David Wagner
University of California, Berkeley

ABSTRACT

Neural networks provide state-of-the-art results for most machine learning tasks. Unfortunately, neural networks are vulnerable to adversarial examples: given an input x and any target classification t, it is possible to find a new input x' that is similar to x but classified as t. This makes it difficult to apply neural networks in security-critical areas. Defensive distillation is a recently proposed approach that can take an arbitrary neural network, and increase its robustness, reducing the success rate of current attacks' ability to find adversarial examples from 95% to 0.5%.

In this paper, we demonstrate that defensive distillation does not significantly increase the robustness of neural networks by introducing three new attack algorithms that are successful on both distilled and undistilled neural networks with 100% probability. Our attacks are tailored to three distance metrics used previously in the literature, and when compared to previous adversarial example generation algorithms, our attacks are often much more effective (and never worse). Furthermore, we propose using high-confidence adversarial examples in a simple transferability test we show can also be used to break defensive distillation. We hope our attacks will be used as a benchmark in future defense attempts to create neural networks that resist adversarial examples.

I. INTRODUCTION

Deep neural networks have become increasingly effective at many difficult machine-learning tasks. In the image recognition domain, they are able to recognize images with near-human accuracy [27], [25]. They are also used for speech recognition [18], natural language processing [1], and playing games [43], [32].

However, researchers have discovered that existing neural networks are vulnerable to attack. Szegedy *et al.* [46] first noticed the existence of *adversarial examples* in the image classification domain: it is possible to transform an image by a small amount and thereby change how the image is classified. Often, the total amount of change required can be so small as to be undetectable.

The degree to which attackers can find adversarial examples limits the domains in which neural networks can be used. For example, if we use neural networks in self-driving cars, adversarial examples could allow an attacker to cause the car to take unwanted actions.

The existence of adversarial examples has inspired research on how to harden neural networks against these kinds of attacks. Many early attempts to secure neural networks failed or provided only marginal robustness improvements [15], [2], [20], [42].

Fig. 1. An illustration of our attacks on a defensively distilled network. The leftmost column contains the starting image. The next three columns show adversarial examples generated by our L_2, L_∞, and L_0 algorithms, respectively. All images start out classified correctly with label l, and the three misclassified instances share the same misclassified label of $l+1 \pmod{10}$. Images were chosen as the first of their class from the test set.

Defensive distillation [39] is one such recent defense proposed for hardening neural networks against adversarial examples. Initial analysis proved to be very promising: defensive distillation defeats existing attack algorithms and reduces their success probability from 95% to 0.5%. Defensive distillation can be applied to any feed-forward neural network and only requires a single re-training step, and is currently one of the only defenses giving strong security guarantees against adversarial examples.

In general, there are two different approaches one can take to evaluate the robustness of a neural network: attempt to prove a lower bound, or construct attacks that demonstrate an upper bound. The former approach, while sound, is substantially more difficult to implement in practice, and all attempts have required approximations [2], [21]. On the other hand, if the

© 2017, Nicholas Carlini. Under license to IEEE.
DOI 10.1109/SP.2017.49

IEEE computer society

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

attacks used in the the latter approach are not sufficiently strong and fail often, the upper bound may not be useful.

In this paper we create a set of attacks that can be used to construct an upper bound on the robustness of neural networks. As a case study, we use these attacks to demonstrate that defensive distillation does not actually eliminate adversarial examples. We construct three new attacks (under three previously used distance metrics: L_0, L_2, and L_∞) that succeed in finding adversarial examples for 100% of images on defensively distilled networks. While defensive distillation stops previously published attacks, it cannot resist the more powerful attack techniques we introduce in this paper.

This case study illustrates the general need for better techniques to evaluate the robustness of neural networks: while distillation was shown to be secure against the current state-of-the-art attacks, it fails against our stronger attacks. Furthermore, when comparing our attacks against the current state-of-the-art on standard unsecured models, our methods generate adversarial examples with less total distortion in every case. We suggest that our attacks are a better baseline for evaluating candidate defenses: before placing any faith in a new possible defense, we suggest that designers at least check whether it can resist our attacks.

We additionally propose using high-confidence adversarial examples to evaluate the robustness of defenses. Transferability [46], [11] is the well-known property that adversarial examples on one model are often also adversarial on another model. We demonstrate that adversarial examples from our attacks are transferable from the unsecured model to the defensively distilled (secured) model. In general, we argue that any defense must demonstrate it is able to break the transferability property.

We evaluate our attacks on three standard datasets: MNIST [28], a digit-recognition task (0-9); CIFAR-10 [24], a small-image recognition task, also with 10 classes; and ImageNet [9], a large-image recognition task with 1000 classes.

Figure 1 shows examples of adversarial examples our techniques generate on defensively distilled networks trained on the MNIST and CIFAR datasets.

In one extreme example for the ImageNet classification task, we can cause the Inception v3 [45] network to incorrectly classify images by changing only the lowest order bit of each pixel. Such changes are impossible to detect visually.

To enable others to more easily use our work to evaluate the robustness of other defenses, all of our adversarial example generation algorithms (along with code to train the models we use, to reproduce the results we present) are available online at http://nicholas.carlini.com/code/nn_robust_attacks.

This paper makes the following contributions:

- We introduce three new attacks for the L_0, L_2, and L_∞ distance metrics. Our attacks are significantly more effective than previous approaches. Our L_0 attack is the first published attack that can cause targeted misclassification on the ImageNet dataset.
- We apply these attacks to defensive distillation and discover that distillation provides little security benefit over un-distilled networks.
- We propose using high-confidence adversarial examples in a simple transferability test to evaluate defenses, and show this test breaks defensive distillation.
- We systematically evaluate the choice of the objective function for finding adversarial examples, and show that the choice can dramatically impact the efficacy of an attack.

II. Background

A. Threat Model

Machine learning is being used in an increasing array of settings to make potentially security critical decisions: self-driving cars [3], [4], drones [10], robots [33], [22], anomaly detection [6], malware classification [8], [40], [48], speech recognition and recognition of voice commands [17], [13], NLP [1], and many more. Consequently, understanding the security properties of deep learning has become a crucial question in this area. The extent to which we can construct adversarial examples influences the settings in which we may want to (or not want to) use neural networks.

In the speech recognition domain, recent work has shown [5] it is possible to generate audio that sounds like speech to machine learning algorithms but not to humans. This can be used to control user's devices without their knowledge. For example, by playing a video with a hidden voice command, it may be possible to cause a smart phone to visit a malicious webpage to cause a drive-by download. This work focused on conventional techniques (Gaussian Mixture Models and Hidden Markov Models), but as speech recognition is increasingly using neural networks, the study of adversarial examples becomes relevant in this domain. [1]

In the space of malware classification, the existence of adversarial examples not only limits their potential application settings, but entirely defeats its purpose: an adversary who is able to make only slight modifications to a malware file that cause it to remain malware, but become classified as benign, has entirely defeated the malware classifier [8], [14].

Turning back to the threat to self-driving cars introduced earlier, this is not an unrealistic attack: it has been shown that adversarial examples are possible in the physical world [26] after taking pictures of them.

The key question then becomes exactly how much distortion we must add to cause the classification to change. In each domain, the distance metric that we must use is different. In the space of images, which we focus on in this paper, we rely on previous work that suggests that various L_p norms are reasonable approximations of human perceptual distance (see Section II-D for more information).

We assume in this paper that the adversary has complete access to a neural network, including the architecture and all paramaters, and can use this in a white-box manner. This is a conservative and realistic assumption: prior work has shown it

[1] Strictly speaking, hidden voice commands are not adversarial examples because they are not similar to the original input [5].

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

is possible to train a substitute model given black-box access to a target model, and by attacking the substitute model, we can then transfer these attacks to the target model. [37]

Given these threats, there have been various attempts [15], [2], [20], [42], [39] at constructing defenses that increase the *robustness* of a neural network, defined as a measure of how easy it is to find adversarial examples that are close to their original input.

In this paper we study one of these, *distillation as a defense* [39], that hopes to secure an arbitrary neural network. This type of defensive distillation was shown to make generating adversarial examples nearly impossible for existing attack techniques [39]. We find that although the current state-of-the-art fails to find adversarial examples for defensively distilled networks, the stronger attacks we develop in this paper *are* able to construct adversarial examples.

B. Neural Networks and Notation

A neural network is a function $F(x) = y$ that accepts an input $x \in \mathbb{R}^n$ and produces an output $y \in \mathbb{R}^m$. The model F also implicitly depends on some model parameters θ; in our work the model is fixed, so for convenience we don't show the dependence on θ.

In this paper we focus on neural networks used as an m-class classifier. The output of the network is computed using the softmax function, which ensures that the output vector y satisfies $0 \leq y_i \leq 1$ and $y_1 + \cdots + y_m = 1$. The output vector y is thus treated as a probability distribution, i.e., y_i is treated as the probability that input x has class i. The classifier assigns the label $C(x) = \arg\max_i F(x)_i$ to the input x. Let $C^*(x)$ be the correct label of x. The inputs to the softmax function are called *logits*.

We use the notation from Papernot et al. [39]: define F to be the full neural network including the softmax function, $Z(x) = z$ to be the output of all layers except the softmax (so z are the logits), and

$$F(x) = \text{softmax}(Z(x)) = y.$$

A neural network typically [2] consists of layers

$$F = \text{softmax} \circ F_n \circ F_{n-1} \circ \cdots \circ F_1$$

where

$$F_i(x) = \sigma(\theta_i \cdot x) + \hat{\theta}_i$$

for some non-linear activation function σ, some matrix θ_i of model weights, and some vector $\hat{\theta}_i$ of model biases. Together θ and $\hat{\theta}$ make up the model parameters. Common choices of σ are tanh [31], sigmoid, ReLU [29], or ELU [7]. In this paper we focus primarily on networks that use a ReLU activation function, as it currently is the most widely used activation function [45], [44], [31], [39].

We use image classification as our primary evaluation domain. An $h \times w$-pixel grey-scale image is a two-dimensional vector $x \in \mathbb{R}^{hw}$, where x_i denotes the intensity of pixel i and is scaled to be in the range $[0, 1]$. A color RGB image is a three-dimensional vector $x \in \mathbb{R}^{3hw}$. We do not convert RGB images to HSV, HSL, or other cylindrical coordinate representations of color images: the neural networks act on raw pixel values.

[2]Most simple networks have this simple linear structure, however other more sophisticated networks have more complicated structures (e.g., ResNet [16] and Inception [45]). The network architecture does not impact our attacks.

C. Adversarial Examples

Szegedy *et al.* [46] first pointed out the existence of *adversarial examples*: given a valid input x and a target $t \neq C^*(x)$, it is often possible to find a similar input x' such that $C(x') = t$ yet x, x' are close according to some distance metric. An example x' with this property is known as a *targeted* adversarial example.

A less powerful attack also discussed in the literature instead asks for *untargeted* adversarial examples: instead of classifying x as a given target class, we only search for an input x' so that $C(x') \neq C^*(x)$ and x, x' are close. Untargeted attacks are strictly less powerful than targeted attacks and we do not consider them in this paper. [3]

Instead, we consider three different approaches for how to choose the target class, in a targeted attack:

- *Average Case:* select the target class *uniformly at random* among the labels that are not the correct label.
- *Best Case:* perform the attack against all incorrect classes, and report the target class that was *least difficult* to attack.
- *Worst Case:* perform the attack against all incorrect classes, and report the target class that was *most difficult* to attack.

In all of our evaluations we perform all three types of attacks: best-case, average-case, and worst-case. Notice that if a classifier is only accurate 80% of the time, then the best case attack will require a change of 0 in 20% of cases.

On ImageNet, we approximate the best-case and worst-case attack by sampling 100 random target classes out of the 1,000 possible for efficiency reasons.

D. Distance Metrics

In our definition of adversarial examples, we require use of a distance metric to quantify similarity. There are three widely-used distance metrics in the literature for generating adversarial examples, all of which are L_p norms.

The L_p distance is written $\|x - x'\|_p$, where the p-norm $\|\cdot\|_p$ is defined as

$$\|v\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}}.$$

In more detail:

[3]An untargeted attack is simply a more efficient (and often less accurate) method of running a targeted attack for each target and taking the closest. In this paper we focus on identifying the most accurate attacks, and do not consider untargeted attacks.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

1) L_0 distance measures the number of coordinates i such that $x_i \neq x'_i$. Thus, the L_0 distance corresponds to the number of pixels that have been altered in an image.[4]
   Papernot *et al.* argue for the use of the L_0 distance metric, and it is the primary distance metric under which defensive distillation's security is argued [39].
2) L_2 distance measures the standard Euclidean (root-mean-square) distance between x and x'. The L_2 distance can remain small when there are many small changes to many pixels.
   This distance metric was used in the initial adversarial example work [46].
3) L_∞ distance measures the maximum change to any of the coordinates:

$$\|x - x'\|_\infty = \max(|x_1 - x'_1|, \ldots, |x_n - x'_n|).$$

   For images, we can imagine there is a maximum budget, and each pixel is allowed to be changed by up to this limit, with no limit on the number of pixels that are modified.
   Goodfellow *et al.* argue that L_∞ is the optimal distance metric to use [47] and in a follow-up paper Papernot *et al.* argue distillation is secure under this distance metric [36].

No distance metric is a perfect measure of human perceptual similarity, and we pass no judgement on exactly which distance metric is optimal. We believe constructing and evaluating a good distance metric is an important research question we leave to future work.

However, since most existing work has picked one of these three distance metrics, and since defensive distillation argued security against two of these, we too use these distance metrics and construct attacks that perform superior to the state-of-the-art for each of these distance metrics.

When reporting all numbers in this paper, we report using the distance metric as defined above, on the range $[0, 1]$. (That is, changing a pixel in a greyscale image from full-on to full-off will result in L_2 change of 1.0 and a L_∞ change of 1.0, not 255.)

E. Defensive Distillation

We briefly provide a high-level overview of defensive distillation. We provide a complete description later in Section VIII.

To defensively distill a neural network, begin by first training a network with identical architecture on the training data in a standard manner. When we compute the softmax while training this network, replace it with a more-smooth version of the softmax (by dividing the logits by some constant T). At the end of training, generate the *soft training labels* by evaluating this network on each of the training instances and taking the output labels of the network.

[4]In RGB images, there are three channels that each can change. We count the number of *pixels* that are different, where two pixels are considered different if *any* of the three colors are different. We do not consider a distance metric where an attacker can change one color plane but not another meaningful. We relax this requirement when comparing to other L_0 attacks that do not make this assumption to provide for a fair comparison.

Then, throw out the first network and use only the soft training labels. With those, train a second network where instead of training it on the original training labels, use the soft labels. This trains the second model to behave like the first model, and the soft labels convey additional hidden knowledge learned by the first model.

The key insight here is that by training to match the first network, we will hopefully avoid over-fitting against any of the training data. If the reason that neural networks exist is because neural networks are highly non-linear and have "blind spots" [46] where adversarial examples lie, then preventing this type of over-fitting might remove those blind spots.

In fact, as we will see later, defensive distillation does not remove adversarial examples. One potential reason this may occur is that others [11] have argued the reason adversarial examples exist is not due to blind spots in a highly non-linear neural network, but due only to the locally-linear nature of neural networks. This so-called linearity hypothesis appears to be true [47], and under this explanation it is perhaps less surprising that distillation does not increase the robustness of neural networks.

F. Organization

The remainder of this paper is structured as follows. In the next section, we survey existing attacks that have been proposed in the literature for generating adversarial examples, for the L_2, L_∞, and L_0 distance metrics. We then describe our attack algorithms that target the same three distance metrics and provide superior results to the prior work. Having developed these attacks, we review defensive distillation in more detail and discuss why the existing attacks fail to find adversarial examples on defensively distilled networks. Finally, we attack defensive distillation with our new algorithms and show that it provides only limited value.

III. Attack Algorithms

A. L-BFGS

Szegedy *et al.* [46] generated adversarial examples using box-constrained L-BFGS. Given an image x, their method finds a different image x' that is similar to x under L_2 distance, yet is labeled differently by the classifier. They model the problem as a constrained minimization problem:

$$\begin{aligned} \text{minimize} \quad & \|x - x'\|_2^2 \\ \text{such that} \quad & C(x') = l \\ & x' \in [0,1]^n \end{aligned}$$

This problem can be very difficult to solve, however, so Szegedy *et al.* instead solve the following problem:

$$\begin{aligned} \text{minimize} \quad & c \cdot \|x - x'\|_2^2 + \text{loss}_{F,l}(x') \\ \text{such that} \quad & x' \in [0,1]^n \end{aligned}$$

where $\text{loss}_{F,l}$ is a function mapping an image to a positive real number. One common loss function to use is cross-entropy. Line search is performed to find the constant $c > 0$ that yields an adversarial example of minimum distance: in other words,

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

we repeatedly solve this optimization problem for multiple values of c, adaptively updating c using bisection search or any other method for one-dimensional optimization.

B. Fast Gradient Sign

The fast gradient sign [11] method has two key differences from the L-BFGS method: first, it is optimized for the L_∞ distance metric, and second, it is designed primarily to be fast instead of producing very close adversarial examples. Given an image x the fast gradient sign method sets

$$x' = x - \epsilon \cdot \text{sign}(\nabla \text{loss}_{F,t}(x)),$$

where ϵ is chosen to be sufficiently small so as to be undetectable, and t is the target label. Intuitively, for each pixel, the fast gradient sign method uses the gradient of the loss function to determine in which direction the pixel's intensity should be changed (whether it should be increased or decreased) to minimize the loss function; then, it shifts all pixels simultaneously.

It is important to note that the fast gradient sign attack was designed to be *fast*, rather than optimal. It is not meant to produce the minimal adversarial perturbations.

Iterative Gradient Sign: Kurakin *et al.* introduce a simple refinement of the fast gradient sign method [26] where instead of taking a single step of size ϵ in the direction of the gradient-sign, multiple smaller steps α are taken, and the result is clipped by the same ϵ. Specifically, begin by setting

$$x'_0 = 0$$

and then on each iteration

$$x'_i = x'_{i-1} - \text{clip}_\epsilon(\alpha \cdot \text{sign}(\nabla \text{loss}_{F,t}(x'_{i-1})))$$

Iterative gradient sign was found to produce superior results to fast gradient sign [26].

C. JSMA

Papernot *et al.* introduced an attack optimized under L_0 distance [38] known as the Jacobian-based Saliency Map Attack (JSMA). We give a brief summary of their attack algorithm; for a complete description and motivation, we encourage the reader to read their original paper [38].

At a high level, the attack is a greedy algorithm that picks pixels to modify one at a time, increasing the target classification on each iteration. They use the gradient $\nabla Z(x)_l$ to compute a *saliency map*, which models the impact each pixel has on the resulting classification. A large value indicates that changing it will significantly increase the likelihood of the model labeling the image as the target class l. Given the saliency map, it picks the most important pixel and modify it to increase the likelihood of class l. This is repeated until either more than a set threshold of pixels are modified which makes the attack detectable, or it succeeds in changing the classification.

In more detail, we begin by defining the saliency map in terms of a pair of pixels p, q. Define

$$\alpha_{pq} = \sum_{i \in \{p,q\}} \frac{\partial Z(x)_t}{\partial x_i}$$

$$\beta_{pq} = \left(\sum_{i \in \{p,q\}} \sum_j \frac{\partial Z(x)_j}{\partial x_i} \right) - \alpha_{pq}$$

so that α_{pq} represents how much changing both pixels p and q will change the target classification, and β_{pq} represents how much changing p and q will change all other outputs. Then the algorithm picks

$$(p^*, q^*) = \arg\max_{(p,q)} (-\alpha_{pq} \cdot \beta_{pq}) \cdot (\alpha_{pq} > 0) \cdot (\beta_{pq} < 0)$$

so that $\alpha_{pq} > 0$ (the target class is more likely), $\beta_{pq} < 0$ (the other classes become less likely), and $-\alpha_{pq} \cdot \beta_{pq}$ is largest.

Notice that JSMA uses the output of the second-to-last layer Z, the logits, in the calculation of the gradient: the output of the softmax F is *not* used. We refer to this as the **JSMA-Z** attack.

However, when the authors apply this attack to their defensively distilled networks, they modify the attack so it uses F instead of Z. In other words, their computation uses the output of the softmax (F) instead of the logits (Z). We refer to this modification as the **JSMA-F** attack.[5]

When an image has multiple color channels (e.g., RGB), this attack considers the L_0 difference to be 1 for each color channel changed independently (so that if all three color channels of one pixel change change, the L_0 norm would be 3). While we do not believe this is a meaningful threat model, when comparing to this attack, we evaluate under both models.

D. Deepfool

Deepfool [34] is an untargeted attack technique optimized for the L_2 distance metric. It is efficient and produces closer adversarial examples than the L-BFGS approach discussed earlier.

The authors construct Deepfool by imagining that the neural networks are totally linear, with a hyperplane separating each class from another. From this, they analytically derive the optimal solution to this simplified problem, and construct the adversarial example.

Then, since neural networks are not actually linear, they take a step towards that solution, and repeat the process a second time. The search terminates when a true adversarial example is found.

The exact formulation used is rather sophisticated; interested readers should refer to the original work [34].

IV. Experimental Setup

Before we develop our attack algorithms to break distillation, we describe how we train the models on which we will evaluate our attacks.

[5]We verified this via personal communication with the authors.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

Layer Type	MNIST Model	CIFAR Model
Convolution + ReLU	3×3×32	3×3×64
Convolution + ReLU	3×3×32	3×3×64
Max Pooling	2×2	2×2
Convolution + ReLU	3×3×64	3×3×128
Convolution + ReLU	3×3×64	3×3×128
Max Pooling	2×2	2×2
Fully Connected + ReLU	200	256
Fully Connected + ReLU	200	256
Softmax	10	10

TABLE I
MODEL ARCHITECTURES FOR THE MNIST AND CIFAR MODELS. THIS ARCHITECTURE IS IDENTICAL TO THAT OF THE ORIGINAL DEFENSIVE DISTILLATION WORK. [39]

Parameter	MNIST Model	CIFAR Model
Learning Rate	0.1	0.01 (decay 0.5)
Momentum	0.9	0.9 (decay 0.5)
Delay Rate	-	10 epochs
Dropout	0.5	0.5
Batch Size	128	128
Epochs	50	50

TABLE II
MODEL PARAMETERS FOR THE MNIST AND CIFAR MODELS. THESE PARAMETERS ARE IDENTICAL TO THAT OF THE ORIGINAL DEFENSIVE DISTILLATION WORK. [39]

We train two networks for the MNIST [28] and CIFAR-10 [24] classification tasks, and use one pre-trained network for the ImageNet classification task [41]. Our models and training approaches are identical to those presented in [39]. We achieve 99.5% accuracy on MNIST, comparable to the state of the art. On CIFAR-10, we achieve 80% accuracy, identical to the accuracy given in the distillation work. [6]

MNIST and CIFAR-10. The model architecture is given in Table I and the hyperparameters selected in Table II. We use a momentum-based SGD optimizer during training.

The CIFAR-10 model significantly overfits the training data even with dropout: we obtain a final training cross-entropy loss of 0.05 with accuracy 98%, compared to a validation loss of 1.2 with validation accuracy 80%. We do not alter the network by performing image augmentation or adding additional dropout as that was not done in [39].

ImageNet. Along with considering MNIST and CIFAR, which are both relatively small datasets, we also consider the ImageNet dataset. Instead of training our own ImageNet model, we use the pre-trained Inception v3 network [45], which achieves 96% top-5 accuracy (that is, the probability that the correct class is one of the five most likely as reported by the network is 96%). Inception takes images as $299\times299\times3$ dimensional vectors.

[6]This is compared to the state-of-the-art result of 95% [12], [44], [31]. However, in order to provide the most accurate comparison to the original work, we feel it is important to reproduce their model architectures.

V. OUR APPROACH

We now turn to our approach for constructing adversarial examples. To begin, we rely on the initial formulation of adversarial examples [46] and formally define the problem of finding an adversarial instance for an image x as follows:

$$\begin{aligned} \text{minimize} \quad & \mathcal{D}(x, x+\delta) \\ \text{such that} \quad & C(x+\delta) = t \\ & x+\delta \in [0,1]^n \end{aligned}$$

where x is fixed, and the goal is to find δ that minimizes $\mathcal{D}(x, x+\delta)$. That is, we want to find some small change δ that we can make to an image x that will change its classification, but so that the result is still a valid image. Here $\mathcal{D}$ is some distance metric; for us, it will be either L_0, L_2, or L_∞ as discussed earlier.

We solve this problem by formulating it as an appropriate optimization instance that can be solved by existing optimization algorithms. There are many possible ways to do this; we explore the space of formulations and empirically identify which ones lead to the most effective attacks.

A. Objective Function

The above formulation is difficult for existing algorithms to solve directly, as the constraint $C(x+\delta) = t$ is highly non-linear. Therefore, we express it in a different form that is better suited for optimization. We define an objective function f such that $C(x+\delta) = t$ if and only if $f(x+\delta) \leq 0$. There are many possible choices for f:

$$\begin{aligned} f_1(x') &= -\text{loss}_{F,t}(x') + 1 \\ f_2(x') &= (\max_{i \neq t}(F(x')_i) - F(x')_t)^+ \\ f_3(x') &= \text{softplus}(\max_{i \neq t}(F(x')_i) - F(x')_t) - \log(2) \\ f_4(x') &= (0.5 - F(x')_t)^+ \\ f_5(x') &= -\log(2F(x')_t - 2) \\ f_6(x') &= (\max_{i \neq t}(Z(x')_i) - Z(x')_t)^+ \\ f_7(x') &= \text{softplus}(\max_{i \neq t}(Z(x')_i) - Z(x')_t) - \log(2) \end{aligned}$$

where s is the correct classification, $(e)^+$ is short-hand for $\max(e, 0)$, $\text{softplus}(x) = \log(1+\exp(x))$, and $\text{loss}_{F,s}(x)$ is the cross entropy loss for x.

Notice that we have adjusted some of the above formula by adding a constant; we have done this only so that the function respects our definition. This does not impact the final result, as it just scales the minimization function.

Now, instead of formulating the problem as

$$\begin{aligned} \text{minimize} \quad & \mathcal{D}(x, x+\delta) \\ \text{such that} \quad & f(x+\delta) \leq 0 \\ & x+\delta \in [0,1]^n \end{aligned}$$

we use the alternative formulation:

$$\begin{aligned} \text{minimize} \quad & \mathcal{D}(x, x+\delta) + c \cdot f(x+\delta) \\ \text{such that} \quad & x+\delta \in [0,1]^n \end{aligned}$$

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

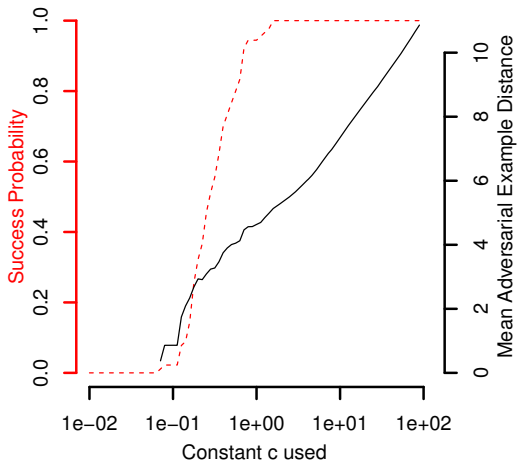

Fig. 2. Sensitivity on the constant c. We plot the L_2 distance of the adversarial example computed by gradient descent as a function of c, for objective function f_6. When $c < .1$, the attack rarely succeeds. After $c > 1$, the attack becomes less effective, but always succeeds.

where $c > 0$ is a suitably chosen constant. These two are equivalent, in the sense that there exists $c > 0$ such that the optimal solution to the latter matches the optimal solution to the former. After instantiating the distance metric $\mathcal{D}$ with an l_p norm, the problem becomes: given x, find δ that solves

$$\begin{aligned} &\text{minimize} \quad \|\delta\|_p + c \cdot f(x+\delta) \\ &\text{such that} \quad x + \delta \in [0,1]^n \end{aligned}$$

Choosing the constant c.

Empirically, we have found that often the best way to choose c is to use the smallest value of c for which the resulting solution x^* has $f(x^*) \leq 0$. This causes gradient descent to minimize both of the terms simultaneously instead of picking only one to optimize over first.

We verify this by running our f_6 formulation (which we found most effective) for values of c spaced uniformly (on a log scale) from $c = 0.01$ to $c = 100$ on the MNIST dataset. We plot this line in Figure 2. [7]

Further, we have found that if choose the smallest c such that $f(x^*) \leq 0$, the solution is within 5% of optimal 70% of the time, and within 30% of optimal 98% of the time, where "optimal" refers to the solution found using the best value of c. Therefore, in our implementations we use modified binary search to choose c.

[7]The corresponding figures for other objective functions are similar; we omit them for brevity.

B. Box constraints

To ensure the modification yields a valid image, we have a constraint on δ: we must have $0 \leq x_i + \delta_i \leq 1$ for all i. In the optimization literature, this is known as a "box constraint." Previous work uses a particular optimization algorithm, L-BFGS-B, which supports box constraints natively.

We investigate three different methods of approaching this problem.

1) *Projected gradient descent* performs one step of standard gradient descent, and then clips all the coordinates to be within the box.
   This approach can work poorly for gradient descent approaches that have a complicated update step (for example, those with momentum): when we clip the actual x_i, we unexpectedly change the input to the next iteration of the algorithm.
2) *Clipped gradient descent* does not clip x_i on each iteration; rather, it incorporates the clipping into the objective function to be minimized. In other words, we replace $f(x+\delta)$ with $f(\min(\max(x+\delta, 0), 1))$, with the min and max taken component-wise.
   While solving the main issue with projected gradient descent, clipping introduces a new problem: the algorithm can get stuck in a flat spot where it has increased some component x_i to be substantially larger than the maximum allowed. When this happens, the partial derivative becomes zero, so even if some improvement is possible by later reducing x_i, gradient descent has no way to detect this.
3) *Change of variables* introduces a new variable w and instead of optimizing over the variable δ defined above, we apply a change-of-variables and optimize over w, setting
   $$\delta_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i.$$
   Since $-1 \leq \tanh(w_i) \leq 1$, it follows that $0 \leq x_i + \delta_i \leq 1$, so the solution will automatically be valid. [8]
   We can think of this approach as a smoothing of clipped gradient descent that eliminates the problem of getting stuck in extreme regions.

These methods allow us to use other optimization algorithms that don't natively support box constraints. We use the Adam [23] optimizer almost exclusively, as we have found it to be the most effective at quickly finding adversarial examples. We tried three solvers — standard gradient descent, gradient descent with momentum, and Adam — and all three produced identical-quality solutions. However, Adam converges substantially more quickly than the others.

C. Evaluation of approaches

For each possible objective function $f(\cdot)$ and method to enforce the box constraint, we evaluate the quality of the adversarial examples found.

[8]Instead of scaling by $\frac{1}{2}$ we scale by $\frac{1}{2} + \epsilon$ to avoid dividing by zero.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

	Best Case						Average Case						Worst Case					
	Change of Variable		Clipped Descent		Projected Descent		Change of Variable		Clipped Descent		Projected Descent		Change of Variable		Clipped Descent		Projected Descent	
	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob
f_1	2.46	100%	2.93	100%	2.31	100%	4.35	100%	5.21	100%	4.11	100%	7.76	100%	9.48	100%	7.37	100%
f_2	4.55	80%	3.97	83%	3.49	83%	3.22	44%	8.99	63%	15.06	74%	2.93	18%	10.22	40%	18.90	53%
f_3	4.54	77%	4.07	81%	3.76	82%	3.47	44%	9.55	63%	15.84	74%	3.09	17%	11.91	41%	24.01	59%
f_4	5.01	86%	6.52	100%	7.53	100%	4.03	55%	7.49	71%	7.60	71%	3.55	24%	4.25	35%	4.10	35%
f_5	1.97	100%	2.20	100%	1.94	100%	3.58	100%	4.20	100%	3.47	100%	6.42	100%	7.86	100%	6.12	100%
f_6	1.94	100%	2.18	100%	1.95	100%	3.47	100%	4.11	100%	3.41	100%	6.03	100%	7.50	100%	5.89	100%
f_7	1.96	100%	2.21	100%	1.94	100%	3.53	100%	4.14	100%	3.43	100%	6.20	100%	7.57	100%	5.94	100%

TABLE III

EVALUATION OF ALL COMBINATIONS OF ONE OF THE SEVEN POSSIBLE OBJECTIVE FUNCTIONS WITH ONE OF THE THREE BOX CONSTRAINT ENCODINGS. WE SHOW THE AVERAGE L_2 DISTORTION, THE STANDARD DEVIATION, AND THE SUCCESS PROBABILITY (FRACTION OF INSTANCES FOR WHICH AN ADVERSARIAL EXAMPLE CAN BE FOUND). EVALUATED ON 1000 RANDOM INSTANCES. WHEN THE SUCCESS IS NOT 100%, MEAN IS FOR SUCCESSFUL ATTACKS ONLY.

To choose the optimal c, we perform 20 iterations of binary search over c. For each selected value of c, we run $10,000$ iterations of gradient descent with the Adam optimizer. [9]

The results of this analysis are in Table III. We evaluate the quality of the adversarial examples found on the MNIST and CIFAR datasets. The relative ordering of each objective function is identical between the two datasets, so for brevity we report only results for MNIST.

There is a factor of three difference in quality between the best objective function and the worst. The choice of method for handling box constraints does not impact the quality of results as significantly for the best minimization functions.

In fact, the worst performing objective function, cross entropy loss, is the approach that was most suggested in the literature previously [46], [42].

Why are some loss functions better than others? When $c = 0$, gradient descent will not make any move away from the initial image. However, a large c often causes the initial steps of gradient descent to perform in an overly-greedy manner, only traveling in the direction which can most easily reduce f and ignoring the $\mathcal{D}$ loss — thus causing gradient descent to find sub-optimal solutions.

This means that for loss function f_1 and f_4, there is no good constant c that is useful throughout the duration of the gradient descent search. Since the constant c weights the relative importance of the distance term and the loss term, in order for a fixed constant c to be useful, the relative value of these two terms should remain approximately equal. This is not the case for these two loss functions.

To explain why this is the case, we will have to take a side discussion to analyze how adversarial examples exist. Consider a valid input x and an adversarial example x' on a network.

What does it look like as we linearly interpolate from x to x'? That is, let $y = \alpha x + (1-\alpha)x'$ for $\alpha \in [0, 1]$. It turns out the value of $Z(\cdot)_t$ is mostly linear from the input to the adversarial example, and therefore the $F(\cdot)_t$ is a logistic. We verify this fact empirically by constructing adversarial examples on the first $1,000$ test images on both the MNIST and CIFAR dataset with our approach, and find the Pearson correlation coefficient $r > .9$.

Given this, consider loss function f_4 (the argument for f_1 is similar). In order for the gradient descent attack to make any change initially, the constant c will have to be large enough that

$$\epsilon < c(f_1(x+\epsilon) - f_1(x))$$

or, as $\epsilon \to 0$,

$$1/c < |\nabla f_1(x)|$$

implying that c must be larger than the inverse of the gradient to make progress, but the gradient of f_1 is identical to $F(\cdot)_t$ so will be tiny around the initial image, meaning c will have to be extremely large.

However, as soon as we leave the immediate vicinity of the initial image, the gradient of $\nabla f_1(x+\delta)$ increases at an exponential rate, making the large constant c cause gradient descent to perform in an overly greedy manner.

We verify all of this theory empirically. When we run our attack trying constants chosen from 10^{-10} to 10^{10} the average constant for loss function f_4 was 10^6.

The average gradient of the loss function f_1 around the valid image is 2^{-20} but 2^{-1} at the closest adversarial example. This means c is a million times larger than it has to be, causing the loss function f_4 and f_1 to perform worse than any of the others.

D. Discretization

We model pixel intensities as a (continuous) real number in the range $[0, 1]$. However, in a valid image, each pixel intensity must be a (discrete) integer in the range $\{0, 1, \ldots, 255\}$. This additional requirement is not captured in our formulation. In practice, we ignore the integrality constraints, solve the continuous optimization problem, and then round to the nearest integer: the intensity of the ith pixel becomes $\lfloor 255(x_i + \delta_i) \rceil$.

This rounding will slightly degrade the quality of the adversarial example. If we need to restore the attack quality, we perform greedy search on the lattice defined by the discrete

[9] Adam converges to 95% of optimum within $1,000$ iterations 92% of the time. For completeness we run it for $10,000$ iterations at each step.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

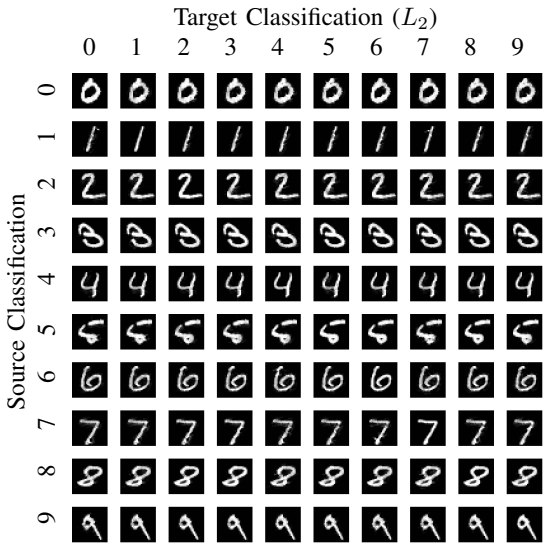

Fig. 3. Our L_2 adversary applied to the MNIST dataset performing a targeted attack for every source/target pair. Each digit is the first image in the dataset with that label.

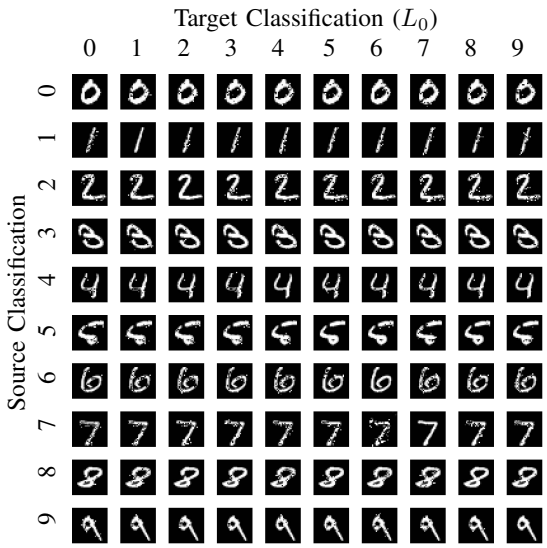

Fig. 4. Our L_0 adversary applied to the MNIST dataset performing a targeted attack for every source/target pair. Each digit is the first image in the dataset with that label.

solutions by changing one pixel value at a time. This greedy search never failed for any of our attacks.

Prior work has largely ignored the integrality constraints.[10] For instance, when using the fast gradient sign attack with $\epsilon = 0.1$ (i.e., changing pixel values by 10%), discretization rarely affects the success rate of the attack. In contrast, in our work, we are able to find attacks that make much smaller changes to the images, so discretization effects cannot be ignored. We take care to always generate valid images; when reporting the success rate of our attacks, they always are for attacks that include the discretization post-processing.

VI. Our Three Attacks

A. Our L_2 Attack

Putting these ideas together, we obtain a method for finding adversarial examples that will have low distortion in the L_2 metric. Given x, we choose a target class t (such that we have $t \neq C^*(x)$) and then search for w that solves

$$\text{minimize} \quad \|\frac{1}{2}(\tanh(w)+1) - x\|_2^2 + c \cdot f(\frac{1}{2}(\tanh(w)+1)$$

with f defined as

$$f(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa).$$

This f is based on the best objective function found earlier, modified slightly so that we can control the confidence with which the misclassification occurs by adjusting κ. The parameter κ encourages the solver to find an adversarial instance x' that will be classified as class t with high confidence. We set $\kappa = 0$ for our attacks but we note here that a side benefit of this formulation is it allows one to control for the desired confidence. This is discussed further in Section VIII-D.

Figure 3 shows this attack applied to our MNIST model for each source digit and target digit. Almost all attacks are visually indistinguishable from the original digit.

A comparable figure (Figure 12) for CIFAR is in the appendix. No attack is visually distinguishable from the baseline image.

Multiple starting-point gradient descent. The main problem with gradient descent is that its greedy search is not guaranteed to find the optimal solution and can become stuck in a local minimum. To remedy this, we pick multiple random starting points close to the original image and run gradient descent from each of those points for a fixed number of iterations. We randomly sample points uniformly from the ball of radius r, where r is the closest adversarial example found so far. Starting from multiple starting points reduces the likelihood that gradient descent gets stuck in a bad local minimum.

B. Our L_0 Attack

The L_0 distance metric is non-differentiable and therefore is ill-suited for standard gradient descent. Instead, we use an iterative algorithm that, in each iteration, identifies some pixels that don't have much effect on the classifier output and then fixes those pixels, so their value will never be changed. The set of fixed pixels grows in each iteration until we have, by process of elimination, identified a minimal (but possibly not minimum) subset of pixels that can be modified to generate an adversarial example. In each iteration, we use our L_2 attack to identify which pixels are unimportant.

In more detail, on each iteration, we call the L_2 adversary, restricted to only modify the pixels in the allowed set. Let

[10]One exception: The JSMA attack [38] handles this by only setting the output value to either 0 or 255.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

δ be the solution returned from the L_2 adversary on input image x, so that $x+\delta$ is an adversarial example. We compute $g = \nabla f(x+\delta)$ (the gradient of the objective function, evaluated at the adversarial instance). We then select the pixel $i = \arg\min_i g_i \cdot \delta_i$ and fix i, i.e., remove i from the allowed set.[11] The intuition is that $g_i \cdot \delta_i$ tells us how much reduction to $f(\cdot)$ we obtain from the ith pixel of the image, when moving from x to $x+\delta$: g_i tells us how much reduction in f we obtain, per unit change to the ith pixel, and we multiply this by how much the ith pixel has changed. This process repeats until the L_2 adversary fails to find an adversarial example.

There is one final detail required to achieve strong results: choosing a constant c to use for the L_2 adversary. To do this, we initially set c to a very low value (e.g., 10^{-4}). We then run our L_2 adversary at this c-value. If it fails, we double c and try again, until it is successful. We abort the search if c exceeds a fixed threshold (e.g., 10^{10}).

JSMA *grows* a set — initially empty — of pixels that are allowed to be changed and sets the pixels to maximize the total loss. In contrast, our attack *shrinks* the set of pixels — initially containing every pixel — that are allowed to be changed.

Our algorithm is significantly more effective than JSMA (see Section VII for an evaluation). It is also efficient: we introduce optimizations that make it about as fast as our L_2 attack with a single starting point on MNIST and CIFAR; it is substantially slower on ImageNet. Instead of starting gradient descent in each iteration from the initial image, we start the gradient descent from the solution found on the previous iteration ("warm-start"). This dramatically reduces the number of rounds of gradient descent needed during each iteration, as the solution with k pixels held constant is often very similar to the solution with $k+1$ pixels held constant.

Figure 4 shows the L_0 attack applied to one digit of each source class, targeting each target class, on the MNIST dataset. The attacks are visually noticeable, implying the L_0 attack is more difficult than L_2. Perhaps the worst case is that of a 7 being made to classify as a 6; interestingly, this attack for L_2 is one of the only visually distinguishable attacks.

A comparable figure (Figure 11) for CIFAR is in the appendix.

C. Our L_∞ Attack

The L_∞ distance metric is not fully differentiable and standard gradient descent does not perform well for it. We experimented with naively optimizing

$$\text{minimize} \quad c \cdot f(x+\delta) + \|\delta\|_\infty$$

However, we found that gradient descent produces very poor results: the $\|\delta\|_\infty$ term only penalizes the largest (in absolute value) entry in δ and has no impact on any of the other. As such, gradient descent very quickly becomes stuck oscillating between two suboptimal solutions. Consider a case where $\delta_i = 0.5$ and $\delta_j = 0.5 - \epsilon$. The L_∞ norm will only penalize δ_i, not δ_j, and $\frac{\partial}{\partial \delta_j}\|\delta\|_\infty$ will be zero at this point. Thus, the gradient imposes no penalty for increasing δ_j, even though it is already large. On the next iteration we might move to a position where δ_j is slightly larger than δ_i, say $\delta_i = 0.5 - \epsilon'$ and $\delta_j = 0.5 + \epsilon''$, a mirror image of where we started. In other words, gradient descent may oscillate back and forth across the line $\delta_i = \delta_j = 0.5$, making it nearly impossible to make progress.

[11] Selecting the index i that minimizes δ_i is simpler, but it yields results with $1.5\times$ higher L_0 distortion.

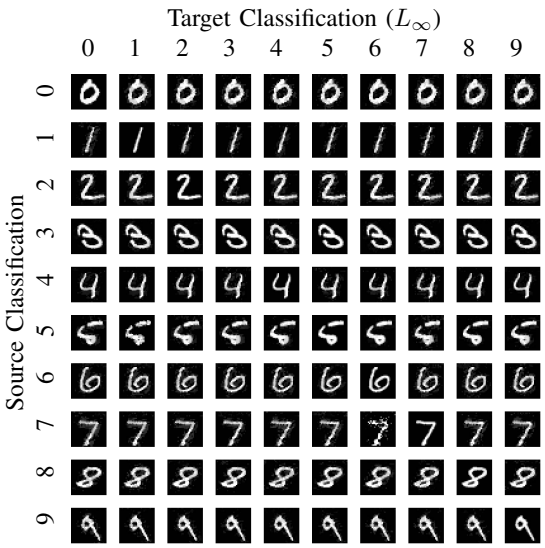

Fig. 5. Our L_∞ adversary applied to the MNIST dataset performing a targeted attack for every source/target pair. Each digit is the first image in the dataset with that label.

We resolve this issue using an iterative attack. We replace the L_2 term in the objective function with a penalty for any terms that exceed τ (initially 1, decreasing in each iteration). This prevents oscillation, as this loss term penalizes all large values simultaneously. Specifically, in each iteration we solve

$$\text{minimize} \quad c \cdot f(x+\delta) + \cdot \sum_i \left[(\delta_i - \tau)^+\right]$$

After each iteration, if $\delta_i < \tau$ for all i, we reduce τ by a factor of 0.9 and repeat; otherwise, we terminate the search.

Again we must choose a good constant c to use for the L_∞ adversary. We take the same approach as we do for the L_0 attack: initially set c to a very low value and run the L_∞ adversary at this c-value. If it fails, we double c and try again, until it is successful. We abort the search if c exceeds a fixed threshold.

Using "warm-start" for gradient descent in each iteration, this algorithm is about as fast as our L_2 algorithm (with a single starting point).

Figure 5 shows the L_∞ attack applied to one digit of each source class, targeting each target class, on the MNSIT dataset. While most differences are not visually noticeable, a few are. Again, the worst case is that of a 7 being made to classify as a 6.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

	Untargeted		Average Case		Least Likely	
	mean	prob	mean	prob	mean	prob
Our L_0	48	100%	410	100%	5200	100%
JSMA-Z	-	0%	-	0%	-	0%
JSMA-F	-	0%	-	0%	-	0%
Our L_2	0.32	100%	0.96	100%	2.22	100%
Deepfool	0.91	100%	-	-	-	-
Our L_∞	0.004	100%	0.006	100%	0.01	100%
FGS	0.004	100%	0.064	2%	-	0%
IGS	0.004	100%	0.01	99%	0.03	98%

TABLE V
COMPARISON OF THE THREE VARIANTS OF TARGETED ATTACK TO PREVIOUS WORK FOR THE INCEPTION V3 MODEL ON IMAGENET. WHEN SUCCESS RATE IS NOT 100%, THE MEAN IS ONLY OVER SUCCESSES.

A comparable figure (Figure 13) for CIFAR is in the appendix. No attack is visually distinguishable from the baseline image.

VII. ATTACK EVALUATION

We compare our targeted attacks to the best results previously reported in prior publications, for each of the three distance metrics.

We re-implement Deepfool, fast gradient sign, and iterative gradient sign. For fast gradient sign, we search over ϵ to find the smallest distance that generates an adversarial example; failures is returned if no ϵ produces the target class. Our iterative gradient sign method is similar: we search over ϵ (fixing $\alpha = \frac{1}{256}$) and return the smallest successful.

For JSMA we use the implementation in CleverHans [35] with only slight modification (we improve performance by $50\times$ with no impact on accuracy).

JSMA is unable to run on ImageNet due to an inherent significant computational cost: recall that JSMA performs search for a pair of pixels p, q that can be changed together that make the target class more likely and other classes less likely. ImageNet represents images as $299 \times 299 \times 3$ vectors, so searching over all pairs of pixels would require 2^{36} work on each step of the calculation. If we remove the search over pairs of pixels, the success of JSMA falls off dramatically. We therefore report it as failing always on ImageNet.

We report success if the attack produced an adversarial example with the correct target label, no matter how much change was required. Failure indicates the case where the attack was entirely unable to succeed.

We evaluate on the first $1,000$ images in the test set on CIFAR and MNSIT. On ImageNet, we report on $1,000$ images that were initially classified correctly by Inception v3 [12]. On ImageNet we approximate the best-case and worst-case results by choosing 100 target classes (10%) at random.

The results are found in Table IV for MNIST and CIFAR, and Table V for ImageNet. [13]

[12]Otherwise the best-case attack results would appear to succeed extremely often artificially low due to the relatively low top-1 accuracy

[13]The complete code to reproduce these tables and figures is available online at http://nicholas.carlini.com/code/nn_robust_attacks.

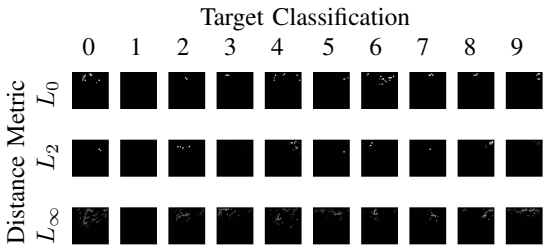

Fig. 6. Targeted attacks for each of the 10 MNIST digits where the starting image is totally black for each of the three distance metrics.

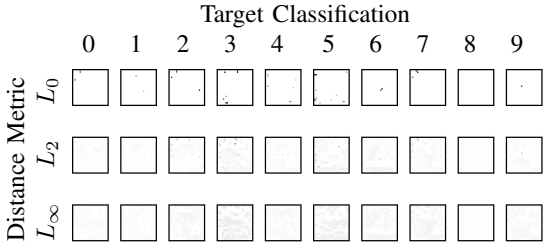

Fig. 7. Targeted attacks for each of the 10 MNIST digits where the starting image is totally white for each of the three distance metrics.

For each distance metric, across all three datasets, our attacks find closer adversarial examples than the previous state-of-the-art attacks, and our attacks never fail to find an adversarial example. Our L_0 and L_2 attacks find adversarial examples with $2\times$ to $10\times$ lower distortion than the best previously published attacks, and succeed with 100% probability. Our L_∞ attacks are comparable in quality to prior work, but their success rate is higher. Our L_∞ attacks on ImageNet are so successful that we can change the classification of an image to any desired label by only flipping the lowest bit of each pixel, a change that would be impossible to detect visually.

As the learning task becomes increasingly more difficult, the previous attacks produce worse results, due to the complexity of the model. In contrast, our attacks perform even better as the task complexity increases. We have found JSMA is unable to find targeted L_0 adversarial examples on ImageNet, whereas ours is able to with 100% success.

It is important to realize that the results between models are not directly comparable. For example, even though a L_0 adversary must change 10 times as many pixels to switch an ImageNet classification compared to a MNIST classification, ImageNet has $114\times$ as many pixels and so the *fraction of pixels* that must change is significantly smaller.

Generating synthetic digits. With our targeted adversary, we can start from *any* image we want and find adversarial examples of each given target. Using this, in Figure 6 we show the minimum perturbation to an entirely-black image required to make it classify as each digit, for each of the distance metrics.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

	Best Case				Average Case				Worst Case			
	MNIST		CIFAR		MNIST		CIFAR		MNIST		CIFAR	
	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob
Our L_0	8.5	100%	5.9	100%	16	100%	13	100%	33	100%	24	100%
JSMA-Z	20	100%	20	100%	56	100%	58	100%	180	98%	150	100%
JSMA-F	17	100%	25	100%	45	100%	110	100%	100	100%	240	100%
Our L_2	1.36	100%	0.17	100%	1.76	100%	0.33	100%	2.60	100%	0.51	100%
Deepfool	2.11	100%	0.85	100%	–	-	–	-	–	-	–	-
Our L_∞	0.13	100%	0.0092	100%	0.16	100%	0.013	100%	0.23	100%	0.019	100%
Fast Gradient Sign	0.22	100%	0.015	99%	0.26	42%	0.029	51%	–	0%	0.34	1%
Iterative Gradient Sign	0.14	100%	0.0078	100%	0.19	100%	0.014	100%	0.26	100%	0.023	100%

TABLE IV
COMPARISON OF THE THREE VARIANTS OF TARGETED ATTACK TO PREVIOUS WORK FOR OUR MNIST AND CIFAR MODELS. WHEN SUCCESS RATE IS NOT 100%, THE MEAN IS ONLY OVER SUCCESSES.

This experiment was performed for the L_0 task previously [38], however when mounting their attack, "for classes 0, 2, 3 and 5 one can clearly recognize the target digit." With our more powerful attacks, none of the digits are recognizable. Figure 7 performs the same analysis starting from an all-white image.

Notice that the all-black image requires no change to become a digit 1 because it is initially classified as a 1, and the all-white image requires no change to become a 8 because the initial image is already an 8.

Runtime Analysis. We believe there are two reasons why one may consider the runtime performance of adversarial example generation algorithms important: first, to understand if the performance would be prohibitive for an adversary to actually mount the attacks, and second, to be used as an inner loop in adversarial re-training [11].

Comparing the exact runtime of attacks can be misleading. For example, we have parallelized the implementation of our L_2 adversary allowing it to run hundreds of attacks simultaneously on a GPU, increasing performance from $10\times$ to $100\times$. However, we did not parallelize our L_0 or L_∞ attacks. Similarly, our implementation of fast gradient sign is parallelized, but JSMA is not. We therefore refrain from giving exact performance numbers because we believe an unfair comparison is worse than no comparison.

All of our attacks, and all previous attacks, are plenty efficient to be used by an adversary. No attack takes longer than a few minutes to run on any given instance.

When compared to L_0, our attacks are $2\times - 10\times$ slower than our optimized JSMA algorithm (and significantly faster than the un-optimized version). Our attacks are typically $10\times - 100\times$ slower than previous attacks for L_2 and L_∞, with exception of iterative gradient sign which we are $10\times$ slower.

VIII. EVALUATING DEFENSIVE DISTILLATION

Distillation was initially proposed as an approach to reduce a large model (the *teacher*) down to a smaller *distilled* model [19]. At a high level, distillation works by first training the teacher model on the training set in a standard manner. Then, we use the teacher to label each instance in the training set with soft labels (the output vector from the teacher network). For example, while the hard label for an image of a hand-written digit 7 will say it is classified as a seven, the soft labels might say it has a 80% chance of being a seven and a 20% chance of being a one. Then, we train the distilled model on the soft labels from the teacher, rather than on the hard labels from the training set. Distillation can potentially increase accuracy on the test set as well as the rate at which the smaller model learns to predict the hard labels [19], [30].

Defensive distillation uses distillation in order to increase the robustness of a neural network, but with two significant changes. First, both the teacher model and the distilled model are identical in size — defensive distillation does not result in smaller models. Second, and more importantly, defensive distillation uses a large *distillation temperature* (described below) to force the distilled model to become more confident in its predictions.

Recall that, the softmax function is the last layer of a neural network. Defensive distillation modifies the softmax function to also include a temperature constant T:

$$\text{softmax}(x, T)_i = \frac{e^{x_i/T}}{\sum_j e^{x_j/T}}$$

It is easy to see that $\text{softmax}(x, T) = \text{softmax}(x/T, 1)$. Intuitively, increasing the temperature causes a "softer" maximum, and decreasing it causes a "harder" maximum. As the limit of the temperature goes to 0, softmax approaches $\max$; as the limit goes to infinity, $\text{softmax}(x)$ approaches a uniform distribution.

Defensive distillation proceeds in four steps:

1) Train a network, the teacher network, by setting the temperature of the softmax to T during the training phase.
2) Compute soft labels by apply the teacher network to each instance in the training set, again evaluating the softmax at temperature T.
3) Train the distilled network (a network with the same shape as the teacher network) on the soft labels, using softmax at temperature T.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

4) Finally, when running the distilled network at test time (to classify new inputs), use temperature 1.

A. Fragility of existing attacks

We briefly investigate the reason that existing attacks fail on distilled networks, and find that existing attacks are very fragile and can easily fail to find adversarial examples even when they exist.

L-BFGS and Deepfool fail due to the fact that the gradient of $F(\cdot)$ is zero almost always, which prohibits the use of the standard objective function.

When we train a distilled network at temperature T and then test it at temperature 1, we effectively cause the inputs to the softmax to become larger by a factor of T. By minimizing the cross entropy during training, the output of the softmax is forced to be close to 1.0 for the correct class and 0.0 for all others. Since $Z(\cdot)$ is divided by T, the distilled network will learn to make the $Z(\cdot)$ values T times larger than they otherwise would be. (Positive values are forced to become about T times larger; negative values are multiplied by a factor of about T and thus become even more negative.) Experimentally, we verified this fact: the mean value of the L_1 norm of $Z(\cdot)$ (the logits) on the undistilled network is 5.8 with standard deviation 6.4; on the distilled network (with $T = 100$), the mean is 482 with standard deviation 457.

Because the values of $Z(\cdot)$ are 100 times larger, when we test at temperature 1, the output of F becomes ϵ in all components except for the output class which has confidence $1-9\epsilon$ for some very small ϵ (for tasks with 10 classes). In fact, in most cases, ϵ is so small that the 32-bit floating-point value is rounded to 0. For similar reasons, the gradient is so small that it becomes 0 when expressed as a 32-bit floating-point value.

This causes the L-BFGS minimization procedure to fail to make progress and terminate. If instead we run L-BFGS with our stable objective function identified earlier, rather than the objective function $\text{loss}_{F,l}(\cdot)$ suggested by Szegedy *et al.* [46], L-BFGS does not fail. An alternate approach to fixing the attack would be to set

$$F'(x) = \text{softmax}(Z(x)/T)$$

where T is the distillation temperature chosen. Then minimizing $\text{loss}_{F',l}(\cdot)$ will not fail, as now the gradients do not vanish due to floating-point arithmetic rounding. This clearly demonstrates the fragility of using the loss function as the objective to minimize.

JSMA-F (whereby we mean the attack uses the output of the final layer $F(\cdot)$) fails for the same reason that L-BFGS fails: the output of the $Z(\cdot)$ layer is very large and so softmax becomes essentially a hard maximum. This is the version of the attack that Papernot *et al.* use to attack defensive distillation in their paper [39].

JSMA-Z (the attack that uses the logits) fails for a completely different reason. Recall that in the $Z(\cdot)$ version of the attack, we use the input to the softmax for computing the gradient instead of the final output of the network. This removes any potential issues with the gradient vanishing, however this introduces new issues. This version of the attack is introduced by Papernot *et al.* [38] but it is not used to attack distillation; we provide here an analysis of why it fails.

Since this attack uses the Z values, it is important to realize the differences in relative impact. If the smallest input to the softmax layer is -100, then, after the softmax layer, the corresponding output becomes practically zero. If this input changes from -100 to -90, the output will still be practically zero. However, if the largest input to the softmax layer is 10, and it changes to 0, this will have a massive impact on the softmax output.

Relating this to parameters used in their attack, α and β represent the size of the change at the input to the softmax layer. It is perhaps surprising that JSMA-Z works on undistilled networks, as it treats all changes as being of equal importance, regardless of how much they change the softmax output. If changing a single pixel would increase the target class by 10, but also increase the least likely class by 15, the attack will not increase that pixel.

Recall that distillation at temperature T causes the value of the logits to be T times larger. In effect, this magnifies the suboptimality noted above as logits that are extremely unlikely but have slight variation can cause the attack to refuse to make any changes.

Fast Gradient Sign fails at first for the same reason L-BFGS fails: the gradients are almost always zero. However, something interesting happens if we attempt the same division trick and divide the logits by T before feeding them to the softmax function: distillation still remains effective [36]. We are unable to explain this phenomenon.

B. Applying Our Attacks

When we apply our attacks to defensively distilled networks, we find distillation provides only marginal value. We re-implement defensive distillation on MNIST and CIFAR-10 as described [39] using the same model we used for our evaluation above. We train our distilled model with temperature $T = 100$, the value found to be most effective [39].

Table VI shows our attacks when applied to distillation. All of the previous attacks fail to find adversarial examples. In contrast, our attack succeeds with 100% success probability for each of the three distance metrics.

When compared to Table IV, distillation has added almost no value: our L_0 and L_2 attacks perform slightly worse, and our L_∞ attack performs approximately equally. All of our attacks succeed with 100% success.

C. Effect of Temperature

In the original work, increasing the temperature was found to consistently reduce attack success rate. On MNIST, this goes from a 91% success rate at $T = 1$ to a 24% success rate for $T = 5$ and finally 0.5% success at $T = 100$.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

	Best Case				Average Case				Worst Case			
	MNIST		CIFAR		MNIST		CIFAR		MNIST		CIFAR	
	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob	mean	prob
Our L_0	10	100%	7.4	100%	19	100%	15	100%	36	100%	29	100%
Our L_2	1.7	100%	0.36	100%	2.2	100%	0.60	100%	2.9	100%	0.92	100%
Our L_∞	0.14	100%	0.002	100%	0.18	100%	0.023	100%	0.25	100%	0.038	100%

TABLE VI
COMPARISON OF OUR ATTACKS WHEN APPLIED TO DEFENSIVELY DISTILLED NETWORKS. COMPARE TO TABLE IV FOR UNDISTILLED NETWORKS.

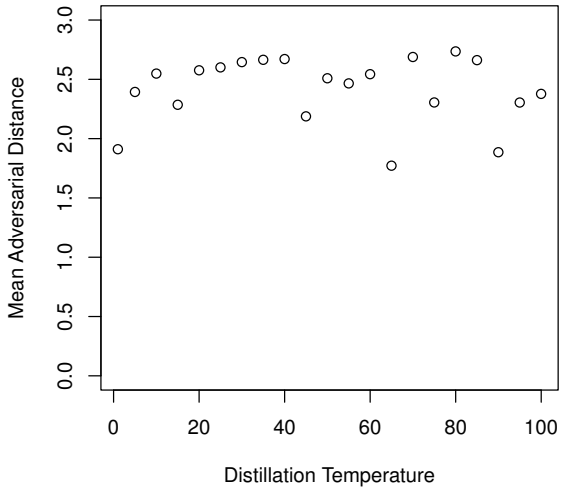

Fig. 8. Mean distance to targeted (with random target) adversarial examples for different distillation temperatures on MNIST. Temperature is uncorrelated with mean adversarial example distance.

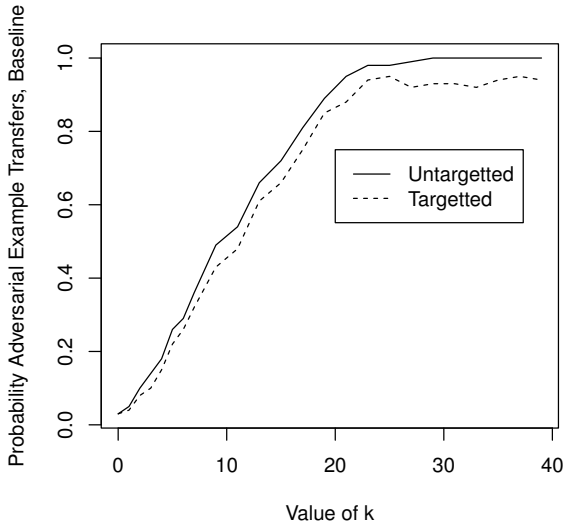

Fig. 9. Probability that adversarial examples transfer from one model to another, for both targeted (the adversarial class remains the same) and untargeted (the image is not the correct class).

We re-implement this experiment with our improved attacks to understand how the choice of temperature impacts robustness. We train models with the temperature varied from $t = 1$ to $t = 100$.

When we re-run our implementation of JSMA, we observe the same effect: attack success rapidly decreases. However, with our improved L_2 attack, we see no effect of temperature on the mean distance to adversarial examples: the correlation coefficient is $\rho = -0.05$. This clearly demonstrates the fact that increasing the distillation temperature does not increase the robustness of the neural network, it only causes existing attacks to fail more often.

D. Transferability

Recent work has shown that an adversarial example for one model will often *transfer* to be an adversarial on a different model, even if they are trained on different sets of training data [46], [11], and even if they use entirely different algorithms (i.e., adversarial examples on neural networks transfer to random forests [37]).

Therefore, any defense that is able to provide robustness against adversarial examples *must* somehow break this transferability property; otherwise, we could run our attack algorithm on an easy-to-attack model, and then transfer those adversarial examples to the hard-to-attack model.

Even though defensive distillation is not robust to our stronger attacks, we demonstrate a second break of distillation by transferring attacks from a standard model to a defensively distilled model.

We accomplish this by finding *high-confidence adversarial examples*, which we define as adversarial examples that are strongly misclassified by the original model. Instead of looking for an adversarial example that just barely changes the classification from the source to the target, we want one where the target is much more likely than any other label.

Recall the loss function defined earlier for L_2 attacks:

$$f(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa).$$

The purpose of the parameter κ is to control the strength of adversarial examples: the larger κ, the stronger the classifi-

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

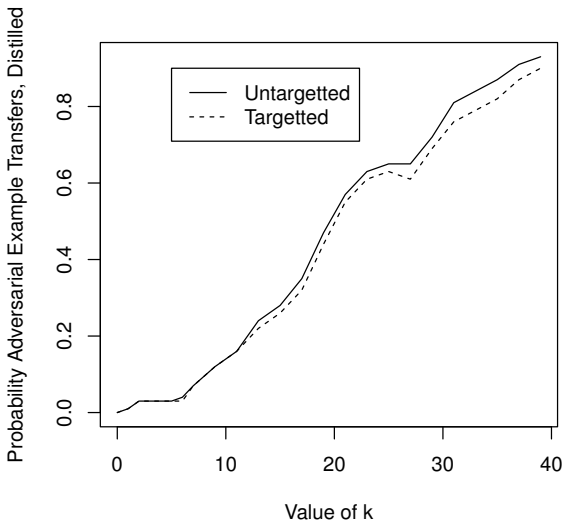

Fig. 10. Probability that adversarial examples transfer from the baseline model to a model trained with defensive distillation at temperature 100.

cation of the adversarial example. This allows us to generate high-confidence adversarial examples by increasing κ.

We first investigate if our hypothesis is true that the stronger the classification on the first model, the more likely it will transfer. We do this by varying κ from 0 to 40.

Our baseline experiment uses two models trained on MNIST as described in Section IV, with each model trained on half of the training data. We find that the transferability success rate increases linearly from $\kappa = 0$ to $\kappa = 20$ and then plateaus at near-100% success for $\kappa \approx 20$, so clearly increasing κ increases the probability of a successful transferable attack.

We then run this same experiment only instead we train the second model with defensive distillation, and find that adversarial examples *do* transfer. This gives us another attack technique for finding adversarial examples on distilled networks.

However, interestingly, the transferability success rate between the unsecured model and the distilled model only reaches 100% success at $\kappa = 40$, in comparison to the previous approach that only required $\kappa = 20$.

We believe that this approach can be used in general to evaluate the robustness of defenses, even if the defense is able to completely block flow of gradients to cause our gradient-descent based approaches from succeeding.

IX. Conclusion

The existence of adversarial examples limits the areas in which deep learning can be applied. It is an open problem to construct defenses that are robust to adversarial examples. In an attempt to solve this problem, defensive distillation was proposed as a general-purpose procedure to increase the robustness of an arbitrary neural network.

In this paper, we propose powerful attacks that defeat defensive distillation, demonstrating that our attacks more generally can be used to evaluate the efficacy of potential defenses. By systematically evaluating many possible attack approaches, we settle on one that can consistently find better adversarial examples than all existing approaches. We use this evaluation as the basis of our three L_0, L_2, and L_∞ attacks.

We encourage those who create defenses to perform the two evaluation approaches we use in this paper:

- **Use a powerful attack** (such as the ones proposed in this paper) to evaluate the robustness of the secured model directly. Since a defense that prevents our L_2 attack will prevent our other attacks, defenders should make sure to establish robustness against the L_2 distance metric.
- **Demonstrate that transferability fails** by constructing high-confidence adversarial examples on a unsecured model and showing they fail to transfer to the secured model.

Acknowledgements

We would like to thank Nicolas Papernot discussing our defensive distillation implementation, and the anonymous reviewers for their helpful feedback. This work was supported by Intel through the ISTC for Secure Computing, Qualcomm, Cisco, the AFOSR under MURI award FA9550-12-1-0040, and the Hewlett Foundation through the Center for Long-Term Cybersecurity.

References

[1] Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., and Collins, M. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042* (2016).

[2] Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., and Criminisi, A. Measuring neural net robustness with constraints. *arXiv preprint arXiv:1605.07262* (2016).

[3] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[4] Bourzac, K. Bringing big neural networks to self-driving cars, smartphones, and drones. http://spectrum.ieee.org/computing/embedded-systems/bringing-big-neural-networks-to-selfdriving-cars-smartphones-and-drones, 2016.

[5] Carlini, N., Mishra, P., Vaidya, T., Zhang, Y., Sherr, M., Shields, C., Wagner, D., and Zhou, W. Hidden voice commands. In *25th USENIX Security Symposium (USENIX Security 16), Austin, TX* (2016).

[6] Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR) 41*, 3 (2009), 15.

[7] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289* (2015).

[8] Dahl, G. E., Stokes, J. W., Deng, L., and Yu, D. Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), IEEE, pp. 3422–3426.

[9] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), IEEE, pp. 248–255.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

[10] GIUSTI, A., GUZZI, J., CIREŞAN, D. C., HE, F.-L., RODRÍGUEZ, J. P., FONTANA, F., FAESSLER, M., FORSTER, C., SCHMIDHUBER, J., DI CARO, G., ET AL. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters 1*, 2 (2016), 661–667.

[11] GOODFELLOW, I. J., SHLENS, J., AND SZEGEDY, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[12] GRAHAM, B. Fractional max-pooling. *arXiv preprint arXiv:1412.6071* (2014).

[13] GRAVES, A., MOHAMED, A.-R., AND HINTON, G. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing* (2013), IEEE, pp. 6645–6649.

[14] GROSSE, K., PAPERNOT, N., MANOHARAN, P., BACKES, M., AND MCDANIEL, P. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435* (2016).

[15] GU, S., AND RIGAZIO, L. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068* (2014).

[16] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.

[17] HINTON, G., DENG, L., YU, D., DAHL, G., RAHMAN MOHAMED, A., JAITLY, N., SENIOR, A., VANHOUCKE, V., NGUYEN, P., SAINATH, T., AND KINGSBURY, B. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine* (2012).

[18] HINTON, G., DENG, L., YU, D., DAHL, G. E., MOHAMED, A.-R., JAITLY, N., SENIOR, A., VANHOUCKE, V., NGUYEN, P., SAINATH, T. N., ET AL. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine 29*, 6 (2012), 82–97.

[19] HINTON, G., VINYALS, O., AND DEAN, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[20] HUANG, R., XU, B., SCHUURMANS, D., AND SZEPESVÁRI, C. Learning with a strong adversary. *CoRR, abs/1511.03034* (2015).

[21] HUANG, X., KWIATKOWSKA, M., WANG, S., AND WU, M. Safety verification of deep neural networks. *arXiv preprint arXiv:1610.06940* (2016).

[22] JANGLOVÁ, D. Neural networks in mobile robot motion. *Cutting Edge Robotics 1*, 1 (2005), 243.

[23] KINGMA, D., AND BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[24] KRIZHEVSKY, A., AND HINTON, G. Learning multiple layers of features from tiny images.

[25] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[26] KURAKIN, A., GOODFELLOW, I., AND BENGIO, S. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[27] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[28] LECUN, Y., CORTES, C., AND BURGES, C. J. The mnist database of handwritten digits, 1998.

[29] MAAS, A. L., HANNUN, A. Y., AND NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML* (2013), vol. 30.

[30] MELICHER, W., UR, B., SEGRETI, S. M., KOMANDURI, S., BAUER, L., CHRISTIN, N., AND CRANOR, L. F. Fast, lean and accurate: Modeling password guessability using neural networks. In *Proceedings of USENIX Security* (2016).

[31] MISHKIN, D., AND MATAS, J. All you need is a good init. *arXiv preprint arXiv:1511.06422* (2015).

[32] MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., AND RIEDMILLER, M. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[33] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., ET AL. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529–533.

[34] MOOSAVI-DEZFOOLI, S.-M., FAWZI, A., AND FROSSARD, P. Deepfool: a simple and accurate method to fool deep neural networks. *arXiv preprint arXiv:1511.04599* (2015).

[35] PAPERNOT, N., GOODFELLOW, I., SHEATSLEY, R., FEINMAN, R., AND MCDANIEL, P. cleverhans v1.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768* (2016).

[36] PAPERNOT, N., AND MCDANIEL, P. On the effectiveness of defensive distillation. *arXiv preprint arXiv:1607.05113* (2016).

[37] PAPERNOT, N., MCDANIEL, P., AND GOODFELLOW, I. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277* (2016).

[38] PAPERNOT, N., MCDANIEL, P., JHA, S., FREDRIKSON, M., CELIK, Z. B., AND SWAMI, A. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (2016), IEEE, pp. 372–387.

[39] PAPERNOT, N., MCDANIEL, P., WU, X., JHA, S., AND SWAMI, A. Distillation as a defense to adversarial perturbations against deep neural networks. *IEEE Symposium on Security and Privacy* (2016).

[40] PASCANU, R., STOKES, J. W., SANOSSIAN, H., MARINESCU, M., AND THOMAS, A. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2015), IEEE, pp. 1916–1920.

[41] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., BERG, A. C., AND FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[42] SHAHAM, U., YAMADA, Y., AND NEGAHBAN, S. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432* (2015).

[43] SILVER, D., HUANG, A., MADDISON, C. J., GUEZ, A., SIFRE, L., VAN DEN DRIESSCHE, G., SCHRITTWIESER, J., ANTONOGLOU, I., PANNEERSHELVAM, V., LANCTOT, M., ET AL. Mastering the game of Go with deep neural networks and tree search. *Nature 529*, 7587 (2016), 484–489.

[44] SPRINGENBERG, J. T., DOSOVITSKIY, A., BROX, T., AND RIEDMILLER, M. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (2014).

[45] SZEGEDY, C., VANHOUCKE, V., IOFFE, S., SHLENS, J., AND WOJNA, Z. Rethinking the Inception architecture for computer vision. *arXiv preprint arXiv:1512.00567* (2015).

[46] SZEGEDY, C., ZAREMBA, W., SUTSKEVER, I., BRUNA, J., ERHAN, D., GOODFELLOW, I., AND FERGUS, R. Intriguing properties of neural networks. *ICLR* (2013).

[47] WARDE-FARLEY, D., AND GOODFELLOW, I. Adversarial perturbations of deep neural networks. *Advanced Structured Prediction, T. Hazan, G. Papandreou, and D. Tarlow, Eds* (2016).

[48] YUAN, Z., LU, Y., WANG, Z., AND XUE, Y. Droid-sec: Deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review* (2014), vol. 44, ACM, pp. 371–372.

APPENDIX

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

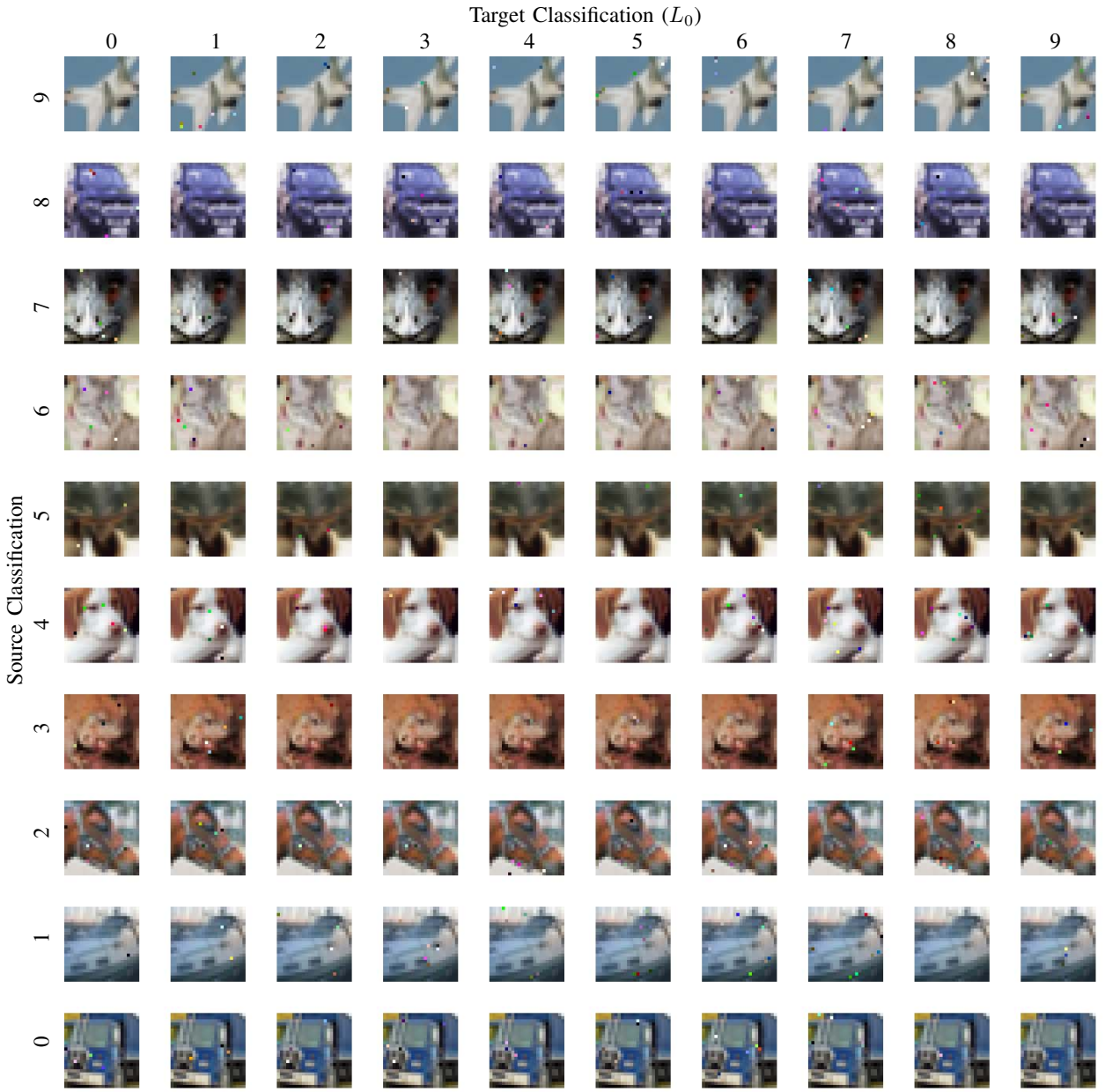

Fig. 11. Our L_0 adversary applied to the CIFAR dataset performing a targeted attack for every source/target pair. Each image is the first image in the dataset with that label.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

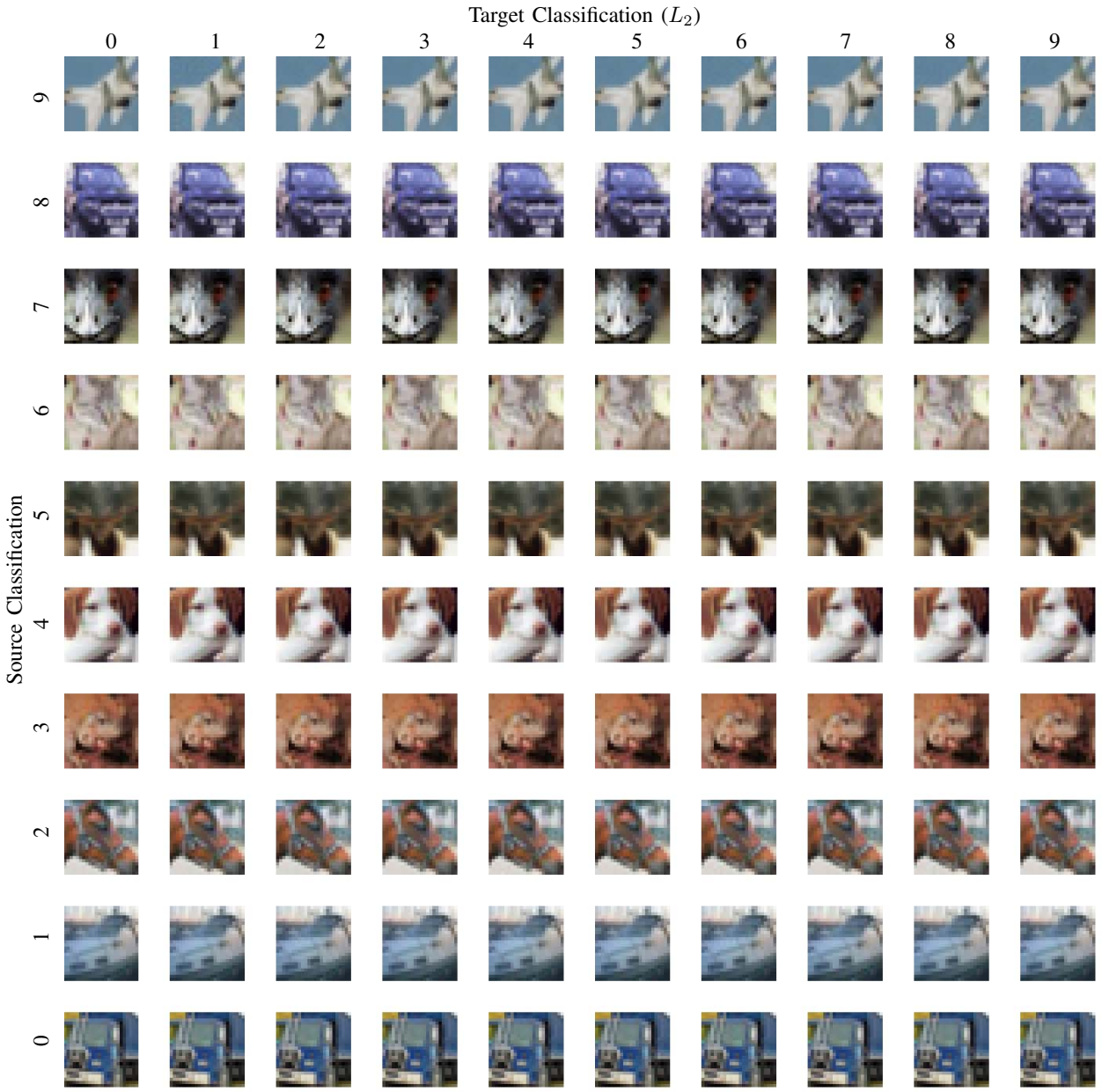

Fig. 12. Our L_2 adversary applied to the CIFAR dataset performing a targeted attack for every source/target pair. Each image is the first image in the dataset with that label.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.

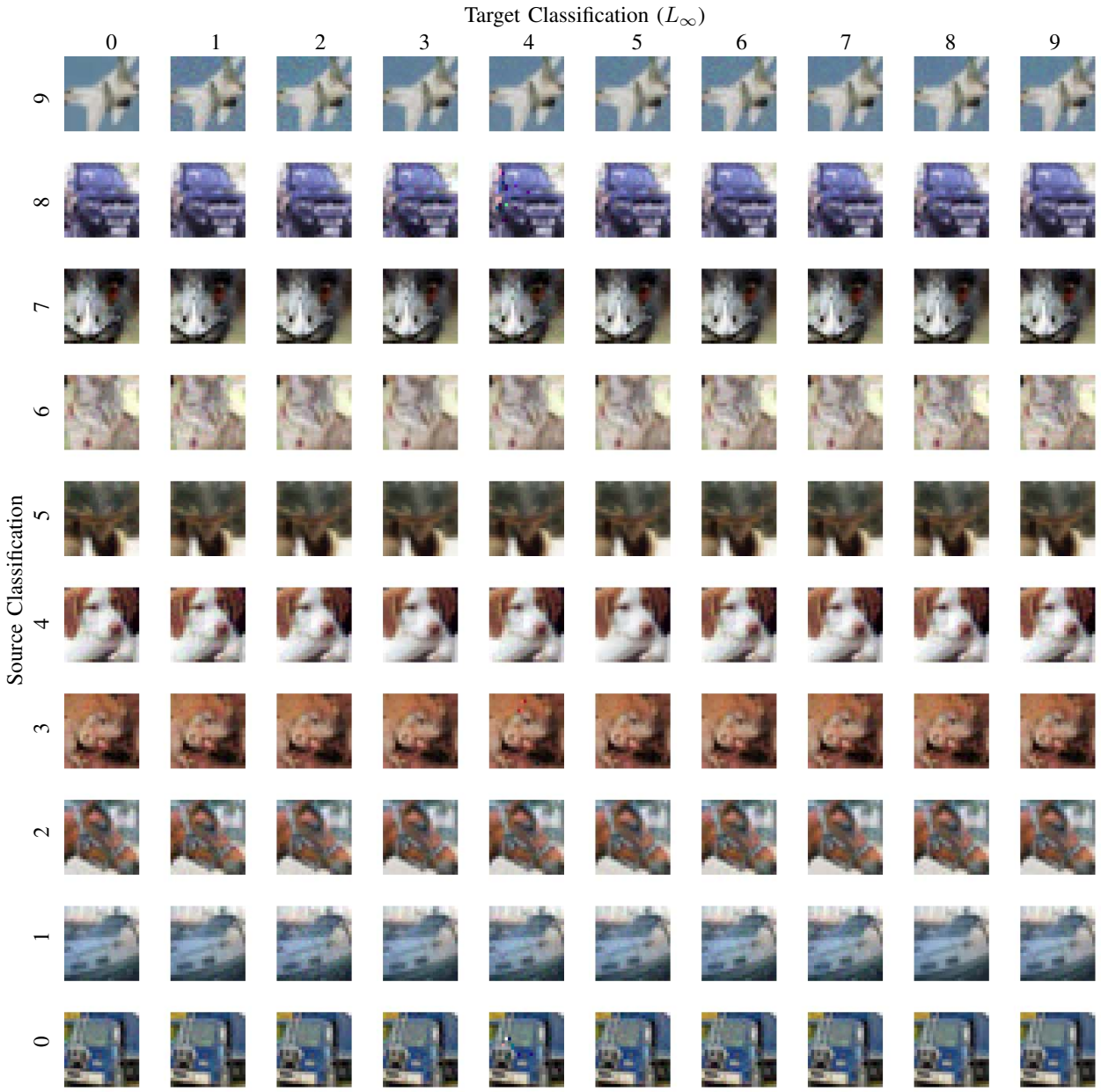

Fig. 13. Our L_∞ adversary applied to the CIFAR dataset performing a targeted attack for every source/target pair. Each image is the first image in the dataset with that label.

Authorized licensed use limited to: IEEE Xplore. Downloaded on May 04,2024 at 21:46:17 UTC from IEEE Xplore. Restrictions apply.