Stochastic Substitute Training: A Gray-box Approach to Craft Adversarial Examples Against Gradient Obfuscation Defenses

Mohammad Hashemi
University of Colorado Boulder
Boulder, Colorado
mohammad.hashemi@colorado.edu

Greg Cusack
University of Colorado Boulder
Boulder, Colorado
gregory.cusack@colorado.edu

Eric Keller
University of Colorado Boulder
Boulder, Colorado
eric.keller@colorado.edu

ABSTRACT

It has been shown that adversaries can craft example inputs to neural networks which are similar to legitimate inputs but have been created to purposely cause the neural network to misclassify the input. These adversarial examples are crafted, for example, by calculating gradients of a carefully defined loss function with respect to the input. As a countermeasure, some researchers have tried to design robust models by blocking or obfuscating gradients, even in white-box settings. Another line of research proposes introducing a separate detector to attempt to detect adversarial examples. This approach also makes use of gradient obfuscation techniques, for example, to prevent the adversary from trying to fool the detector. In this paper, we introduce stochastic substitute training, a gray-box approach that can craft adversarial examples for defenses which obfuscate gradients. For those defenses that have tried to make models more robust, with our technique, an adversary can craft adversarial examples with no knowledge of the defense. For defenses that attempt to detect the adversarial examples, with our technique, an adversary only needs very limited information about the defense to craft adversarial examples. We demonstrate our technique by applying it against two defenses which make models more robust and two defenses which detect adversarial examples.

ACM Reference Format:
Mohammad Hashemi, Greg Cusack, and Eric Keller. 2018. Stochastic Substitute Training: A Gray-box Approach to Craft Adversarial Examples Against Gradient Obfuscation Defenses. In *AISec '18: 11th ACM Workshop on Artificial Intelligence and Security Oct. 19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3270101.3270111

1 INTRODUCTION

Deep learning has evolved in many areas. These deep neural networks show promising results in tasks such as malware detection [37], autonomous driving [7], network intrusion detection [34], diagnosis in medical images[12], and in applications such as image classification, deep neural networks can even surpass human level performance. [10] Deep reinforcement learning has also demonstrated promising results in recent years in many decision making problems, such as human level control in Atari video games [23], defeating the best human players in the game of Go [31], making a humanoid robot run [30], and resource management in a cluster with different resource types [21].

Despite their success in a wide range of applications, deep neural networks, like other traditional classifiers, suffer from a vulnerability to adversarial examples. When working with images, an adversarial example is an image which is carefully modified to make a classifier predict it incorrectly with minimal modifications to the image. In many cases, the perturbation that is added to these images is imperceptible to a human observer. Therefore, a human is likely to classify the images as they did before the alterations. This problem is not limited to modifications to digital images. Kurakin et al. in [15] showed for the first time that this attack is applicable in the physical world as well. Later, Eykholt et al. in [8] showed that an adversary can place a few stickers on a stop sign to fool a classifier, *e.g.,* causing it to predict the sign as a speed limit sign.

Due to the threat that adversarial examples pose, many researchers have proposed solutions to address this vulnerability. These works fall into two main categorizations. In one line of work, researchers introduced different mechanisms to make classifiers more robust to adversarial examples such that the models classify adversarial inputs that are visually close to legitimate inputs correctly [4, 36]. We refer to these defenses as "fortifying defenses". In the other line of work, others have tried to distinguish between legitimate examples and adversarial examples using some detection mechanisms [19, 29]. We refer to these defenses as "detecting defenses". One method an attacker can use to craft an adversarial example is to calculate the gradients of a loss function with respect to the input. Carlini et al. in [5] showed that an adversary can bypass ten detection methods by changing this loss function. Since that time, both the detecting defenses and fortifying defenses have evolved. The defenses now leverage techniques that prevent the adversary from getting a useful gradient from the model or the detector even when the loss function is changed. These techniques are called gradient masking as introduced by Papernot et al. in [26]. Athalye et al. in [1], however, demonstrated these defenses are still vulnerable by showing a white-box approach to craft adversarial examples against these defenses, where the attacker needs to know about the defenses, their parameters, and model parameters.

In this paper, we introduce Stochastic Substitute training (SST), which is an easy and general gray-box attack, for breaking defenses that obfuscate gradients without any knowledge about the model's parameters, the defense parameters, or access to the training dataset. SST only assumes access to the logits (inputs of softmax layer) and doesn't need to be tailored to different defenses in the case where they fortify a model. That is, for fortifying defenses, SST is completely generic. For detecting defenses, the attacker should

AISec '18, October 19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-6004-3/18/10…$15.00
https://doi.org/10.1145/3270101.3270111

bring the detection conditions into the loop of crafting adversarial examples. We do this by training a substitute model with stochastically modified inputs. These inputs are the set of images that an adversary wants to craft adversarial examples for. We evaluate two fortifying defenses and two detecting defenses. But our approach is not limited to these defenses and can be applied to others as well.

We make the following contributions:

- We introduce Stochastic Substitute Training (SST) to craft adversarial examples for models that obfuscate gradients, as old methods, such as those introduced in [6, 9, 27, 33], are ineffective in crafting adversarial examples for models that obfuscate gradients.
- We evaluate two fortifying defenses, random feature nullification (RFN) [36] and thermometer encoding [4], on the MNIST [16] and CIFAR-10 [14] datasets respectively. With SST, we show that an adversary can craft adversarial examples for models fortified with these defenses with no knowledge about the defense and with a small amount of perturbation.
- We evaluate two detecting defenses, SafetyNet [19] and Defense-GAN [29], on the MNIST dataset. We show how an adversary can bypass these detection methods.
- We compare against two black-box attacks [18, 26] that can be used to evaluate defenses which obfuscate gradients without knowledge about the defense. We show that evaluating with these black-box approaches provides the defenses have a false sense of security. Since, with the minimal extra visibility (the logits) in our gray-box approach, we are capable of crafting adversarial examples that the black-box attacks cannot.

The rest of this paper is organized as follows: in Section 2, we provide the reader with background information. Then, in Section 3, we introduce SST, our new approach for crafting adversarial examples for defenses which obfuscate gradients. In Section 4, we evaluate our approach against fortifying defenses, and in section 5, we evaluate our approach against detecting defenses. In Section 6, we compare against two black-box attacks against the aforementioned defenses, and finally in Section 7, we conclude the paper.

2 BACKGROUND

In this section, we first briefly explain how deep neural networks work for image classification and then introduce the notation we use in this paper. Finally, we go over how an adversary can craft an adversarial example.

2.1 Deep Neural Networks

A deep neural network (as a classifier), as illustrated in Figure 1, is a non-linear function which maps an input to a probability vector where each of its elements corresponds to a class score. The element that has the largest score is considered as the prediction. A deep neural network consists of multiple layers that are connected to each other sequentially such that the output of one layer becomes the input of the next layer, and each layer applies a non-linear transformation to its inputs. Each layer has a set of parameters which are initialized randomly. By varying those parameters, the output of the classifier changes. The goal is to find values for the

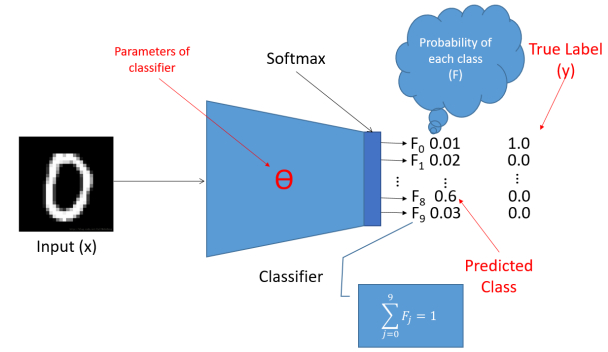

Figure 1: Illustration of a DNN classifier.

parameters such that for most of the inputs, the neural network predicts their labels correctly, which means that the probability corresponding to their true label should be larger than others.

In order to train this network, we want to find the set of parameters that make it predict most of the inputs correctly. So, we have to maximize the score corresponding to the true label for each input. In general, we can say that we need to find a θ that maximizes $\sum_{j=0}^{K} y_j F_j$ for every input. Note that y is a vector and only one of its elements is 1 and the others are 0. Instead of maximizing $\sum_{j=0}^{K} y_j F_j$, we can minimize $-\sum_{j=0}^{K} y_j log(F_j)$. Mathematically, we need to solve the following optimization problem to train a model:

$$argmin_\theta \left(-\frac{1}{N} \sum_{i=0}^{N} \sum_{j=0}^{K} y_{ij} log(F_j(x_i)) \right)$$

where N is the total number of samples in our training set and K is the total number of classes. This is called a cross-entropy loss function, which is a function of θ. A lower value of this function means better predictions over the training set. In order to solve this minimization problem, a technique called gradient descent, or one of its variants such as Adam optimization [13], is used.

2.2 Notation

In the rest of this paper we use the following notation:

- x: the legitimate (clean) input. $x \in [0,1]^m$, where m is the number of pixels in an image.
- y: the label corresponding to the legitimate input.
- x': the adversarial input. $x' \in [0,1]^m$.
- y': the label corresponding to the adversarial input, which is different from its original label.
- y_{target}: the label which an adversary wants to make the classifier output.
- $F(.)$: the classifier which maps an image to a label. For correctly predicted inputs we have $F(x) = y$.
- θ: the parameters of classifier.
- $Z(.)$: the logits, which are inputs of the softmax layer. So, $softmax(Z(x)) = F(x)$.
- δ: the perturbation which is added to a legitimate example to make it adversarial. So, $x' = x + \delta$.

2.3 Adversarial Example

Previous works showed how to craft adversarial examples in white-box and black-box settings [2, 6, 9, 18, 24, 26, 27, 33]. We discuss some of them here.

2.3.1 White-box Setting. Early efforts by Szegedy et al. in [33] and Biggio et al. in [3] showed how to craft an adversarial example. The process for crafting a targeted adversarial example can be reduced to a box-constrained optimization problem as follows:

$$argmin_{\delta}||\delta||_p \text{ s.t. } (x+\delta) \in [0,1]^m \text{ and } F(x+\delta) = y_{target}$$

This optimization problem means that an attacker wants to find the minimum perturbation, so that if she adds it to the input, the classifier would predict it as the attacker's desired target. However, neural networks are not convex, so this optimization problem is intractable and people use different heuristics to find a small enough perturbation that can fool the model. There is another class of attacks, which are known as non-targeted attacks, in which the attacker's goal is to make the classifier misclassify the input to any other label (as opposed to targeted attacks which the goal is to make the classifier output a specific label). For non-targeted attacks the optimization problem can be formulated as follows:

$$argmin_{\delta}||\delta||_p \text{ s.t. } (x+\delta) \in [0,1]^m \text{ and } F(x+\delta) \neq y$$

Carlini et al. in [6] described a way to craft adversarial examples, and we explain it here briefly as we use the same loss function in our attack. They designed their attack by introducing a new objective function. The objective function that they used is as follows:

$$\text{minimize } ||\delta||_p + c.f(x+\delta) \text{ s.t. } x+\delta \in [0,1]^m$$

in which p can be 0,2 or ∞. One of their choices for function f is:

$$f(x') = (max_{i \neq t}(Z(x')_i - Z(x')_t)^+$$

in which Z is the logit which are the inputs to the softmax function, t is the target label, $(e)^+$ is short-hand for $max(e, 0)$, and c is a hyper parameter that determines the trade-off between the amount of distortion and the growth of the target score. As c grows, the amount of distortion and the success probability grows. They showed that by using this function, they can craft adversarial examples for the MNIST, CIFAR-10, and ImageNet datasets with less distortion compared to other white-box attacks. This minimization basically says that we want to find a δ such that its magnitude is minimal (with respect to $l0$, $l2$ or $l\infty$ norm) and the logit value corresponding to the target label is larger than other logits, which makes the classifier predict the input as the target class. This optimization problem is solved by the help of gradient descent, which we mentioned earlier. Carlini and Wagner also showed that they can build adversarial examples that will make the classifier output the target label with higher probability by slightly changing the function f as follows:

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

in which $\kappa >= 0$ and determines the confidence score. By increasing κ, the confidence score of the target class becomes larger. This function basically means that we keep modifying the input as long as $Z(x')_t < max_{i \neq t}(Z(x')_i) + \kappa$.

This technique is not the only way to craft adversarial examples. For more information about crafting adversarial examples in white-box setting we refer the reader to [2, 9, 24, 27, 33].

2.3.2 Black-box setting. Szegedy et al. in [33] also showed that, in many cases, an adversarial example built using one classifier can fool another classifier that has a different architecture and parameters. This property is called the transferability of adversarial examples. By using this property, Carlini et al. in [6] showed that they could craft adversarial examples against classifiers fortified by defensive distillation [28], which block gradients by crafting adversarial examples against a different model with high confidence. Later, Liu et al. in [18] built on top of this idea and crafted adversarial examples against multiple pre-trained models to then be able to transfer them to the target model. It has been also shown by Tramèr et al. in [35] that augmenting training data with adversarial examples generated by a few fixed, pre-trained models significantly improves the robustness of a model in the face of these types of transferable black-box attacks.

Biggio et al. in [3] and Papernot et al. in [26] showed that an adversary can craft adversarial examples against a model in a black-box setting by querying the target model and training a substitute model using the labels predicted by the target model. In this case, after training the substitute model, the adversary can craft adversarial examples against the substitute model in order to transfer them to the target model. Papernot et al. in [26] also showed independently that they can evade defensive distillation by querying the target model in a black-box setting.

Our work is built on top of this transferability property and substitute training approach and provides a better tool for evaluating defenses by considering a more powerful attacker that has access to the logits of the target model.

2.4 Defenses

In general, there are two different approaches for defending against adversarial examples:

- Fortifying Defenses: These types of defenses try to make the classifier predict adversarial examples as their correct class. Techniques that are used include removing adversarial perturbation by transforming the input before feeding it to the classifier, quantization or discretization of the input, and randomization of the input or the model.
- Detecting Defenses: For these types of defenses, the classifier may predict an adversarial example incorrectly, but there is an adversarial detection mechanism that makes the whole model reject those cases. A technique used for these defenses is to augment the classifier with another DNN (or any other model) to classify the input as legitimate or adversarial, or other means of detection such as using some statistics which are assumed to be co-related with adversarial examples.

3 STOCHASTIC SUBSTITUTE TRAINING

In this section, we introduce our new gray-box approach to generating adversarial examples.

3.1 Threat Model

Before describing our approach, it is useful to clarify the threat model. In this paper we consider two different threat models:

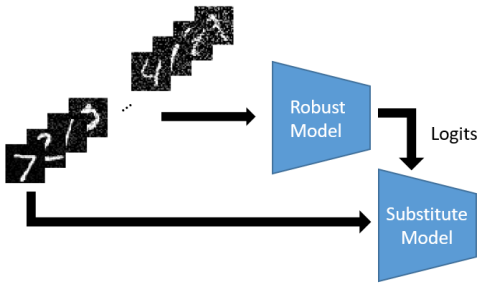

Figure 2: Illustration of Stochastic Substitute Training.

- For evaluating fortifying defenses, we consider an attacker that can send inputs to the model and see the logits. The attacker is not aware that a defense is in place and she doesn't have access to the model or defense parameters.
- For evaluating the defenses that detect adversarial examples, we consider an attacker that knows a detection mechanism is in place. She can send inputs to the model and see logits and the output of the detector but doesn't have access to the model or detector parameters.

3.2 Algorithm

In order to attack the robust classifiers with defenses that obfuscate gradients, we use the transferability property of adversarial examples. In general, we add different levels of random noise to the set of images we want to craft adversarial examples for. In the case of our experiments, this would be the test set of MNIST and CIFAR-10. We then feed this dataset to the robust classifier and record the logits. After that, we train a substitute model with this dataset and the recorded logits. Figure 2 illustrates this process.

For training this model, instead of using a default cross entropy loss, we use the mean square error between the substitute model's logits and logits we got from the robust model as our loss function. More specifically, the loss function is defined as follows:

$$Loss_{SST} = \frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{K}\frac{1}{K}\left(Z_j^{robust}(x_i + r_i) - Z_j^{sub}(x_i + r_i)\right)^2$$

where r_i is the noise added to the sample x_i and N is the total number of inputs in our augmented dataset. Training a substitute model in this way makes the substitute model's decision boundaries for that dataset very close to the robust model, which makes transferability to the robust model easier. Further, since the substitute model is differentiable, we can craft adversarial examples for it using an iterative method. Training a substitute model with images augmented with random noise helps the substitute learn how the robust model's class probabilities change in the neighborhood of each sample. Note that these types of random noises do not necessarily change the prediction of the robust model, but it helps the substitute detect in which directions the correct class score can be decreased. Also, because we assumed that the attacker doesn't know the defense which is in place and how robust the model is, we augment the dataset with different levels of random noise. This is because if we add a low level of random noise, the model might be very robust and the adversarial perturbations found during the crafting procedure may not be sufficient to fool the classifier. On the other hand, if we add a high level of random noise, the model may not be that robust and unnecessary adversarial perturbations would be added to the images during crafting procedure. The reason we use logits instead of class probabilities is that the effect of low level random noise is more obvious in logits compared to probabilities. Using probabilities may not capture the impact of small amount of random noise because of the floating point precision. We empirically found that for complex datasets, training multiple copies of a model with different random noises reduces the required adversarial perturbation on average. We speculate this is because the decision boundaries of the robust model and substitute models are not completely matched, and each of the substitute models approximates the decision boundaries for some specific images better than others.

After training the substitute model for multiple epochs, we craft adversarial examples against it by using the C&W loss function mentioned in Section 2.3 and gradient descent to minimize this loss function iteratively. By doing so, we find a small adversarial perturbation that fools the robust model. We minimize the loss function by using the Adam algorithm as our optimizer. In each iteration, we check if the current perturbation can fool the robust model. If so, we increase the c parameter in the C&W loss function to craft adversarial examples with a smaller amount of distortion in subsequent iterations. We also keep decreasing the value of c in each iteration until we fool the robust model in order to increase the amount of perturbation and chance of transferability. If we couldn't find an adversarial example in the first run, we restart the algorithm and increase κ to build adversarial examples with higher confidence. This might increase the amount of perturbation, but it also increases the chance of transferability to the robust model. Algorithm 1 shows this process for crafting adversarial examples. In this procedure F is the target classifier.

3.3 Benefit of Noisy Data Augmentation

In order to show the benefit of our stochastic substitute training over an approach that uses a substitute model trained without data augmentation, we first trained a model on MNIST as our target model. Then, we trained a substitute model with and without augmenting data with random noises with the first 100 samples from MNIST test set. We made 2100 replication of this set and trained the substitute model on this dataset one epoch with lr=0.001 and another epoch with lr=0.0001. Different levels of random noises were added to different replicas for the one used in SST. Then with each of the substitute models, we crafted an adversarial example with Algorithm 1 for the first 100 samples. The average l2 norm of adversarial examples crafted with stochastic substitute training was 1.57. The average l2 norm for the substitute model which was trained without data augmentation was 3.10. As it can be seen, the average perturbation of images crafted without SST is almost 2 times larger than those which are crafted with SST.

For the sake of comparison, to measure what we are sacrificing by limiting our approach to a gray-box setting, we also crafted adversarial examples with the C&W attack for the target model in

a white-box setting. The average l2 norm of adversarial examples crafted in this way was 1.25, which can be seen as the minimum required perturbation found by current white box techniques to fool the target model for those images. As it can be seen, crafted adversarial examples with C&W are only slightly better than those crafted with SST.

Algorithm 1 Crating adversarial examples

```
procedure CraftAdvExample(x, totalRun, totalIter, F)
    adv ← [0]^m #Adversarial Example
    for each i ∈ [0, totalRun] do
        initialize δ randomly
        for each j ∈ [0, totalIter] do
            take one step of GD using Adam
            x' ← Clip(x + δ)
            if adv == [0]^m then
                decrease c
            end if
            # Check detector's prediction as well (if any)
            if F(x') ≠ y and ||x − x'||_2 < ||x − adv||_2 then
                adv ← x'
                increase c
            end if
        end for
        increase κ
    end for
end procedure
```

4 EVALUATION OF SST AGAINST FORTIFYING DEFENSES

In this section we evaluate the effectiveness of SST against two fortifying defenses – random feature nullification (RFN) [36] and thermometer encoding [4].

4.1 Random Feature Nullification

Wang et al. in [36] proposed an adversary resistant technique to obstruct attackers from constructing impactful adversarial samples. They called this adversarial resistant technique "random feature nullification" and is described as follows:

For each batch of inputs denoted by $X \in R^{n \times m}$, where n is the number of samples and m is the feature vector size, random feature nullification performs element-wise multiplication of X with a randomly generated mask matrix $I_p \in R^{n \times m}$, where its elements are only 1 or 0. The result is then fed to the classifier. During training they generate a mask matrix in a way to randomly select the number of features to nullify and also randomly select which features to nullify. More specifically, for each row of I_p denoted by I_{p^i}, the number of features to be nullified are selected from the Gaussian distribution $N(\mu_p, \sigma_p^2)$, and then a uniform distribution is used for generating each row of I_p. During test-time, the nullification rate is fixed to μ_p, but choosing features in each sample is still random with uniform distribution. The randomness they introduced during test-time prevents an adversary from computing the gradients needed for crafting an adversarial example. In their evaluation they showed that a classifier fortified by RFN can resist against 71.44% of generated adversarial examples in the case where the adversary is allowed to change the value of each pixel by 0.25.

4.1.1 Our Evaluation. Since the authors didn't publish their code, in order to evaluate RFN, we trained a model with the same architecture and parameters they used in their paper. More specifically, the parameters can be found in Table 5 in the Appendix. For the hyper parameters of RFN, we set μ_p to 0.5 and σ to 0.05. During test time for each input, half of its features are nullified before feeding to the DNN. After training the model, we got 96.63% accuracy on the MNIST test set.

For training the substitute model, we added uniform random noise to the test set and created a new data set with 70000 samples. For the first 10000 samples the range of noises was in $[-0.05, 0.05]$, for the next 10000 samples it was in $[-0.1, 0.1]$, and so on. We used Adam optimizer to train the substitute model with 0.001 learning rate for 10 epochs and then 5 epochs with 0.0001 learning rate. It finally reached 97.91% accuracy on legitimate test samples. The substitute model architecture can be found in Table 6 in the Appendix. In this table, the convolution layer parameters are described as $M, K \times K, S$ which refers to a convolution layer with M feature maps, filter size $K \times K$ and stride S. The Max Pooling layer parameters are described as $K \times K, S$, which refers to a Max Pooling layer with pool size $K \times K$ and stride S.

Since RFN is a stochastic defense, feeding the same image to the model multiple times may cause different results. So, an adversarial example may fool the classifier in one run, but it may be predicted correctly in the next run. In the paper, it is not discussed what exactly should be considered as fooling the model. Here we report the accuracy of the model on legitimate samples and the average l2 norm that is required to fool this model in different scenarios. First, we consider an input to be classified correctly if in 100 parallel runs the model can predict it correctly for more than 50 cases. In the second scenario, we change this threshold to 70, and for the last one we change it to 90. For the evaluation of the model, we used the first 100 samples in the test set to generate adversarial examples using our attack. For crafting adversarial examples, we set the learning rate to 0.001. For each sample, we chose the target label as the second most probable class predicted by the robust model. In Algorithm 1, we set the total run to 3 and total iterations to 300. First, we evaluated this model with an l_2 attack. The average l_2 norm for different scenarios can be found in Table 1. The success rate in all cases was 100%. Figure 3 also shows the crafted adversarial examples against this model when the threshold is 50.

For this defense we also generated adversarial examples with $l_\infty = 0.25$. We could generate adversarial examples for 94 samples out of 100 in the first scenario. So, in this case, the resistance rate is only 6%. In the second scenario, we could generate adversarial examples for 92 samples. For the last scenario, we could generate adversarial examples for 75 samples.

4.2 Thermometer Encoding

Buckman et al. in [4] introduced another defense, called thermometer encoding, which makes a model more robust against adversarial examples. It prevents an adversary from calculating the gradients that are needed for crafting adversarial examples. Thermometer

Threshold	Accuracy	L2 norm
50	97.93	2.13
70	96.46	2.43
90	92.78	2.96

Table 1: The accuracy of a model fortified with RFN and the average l2 norm of the required perturbation across different thresholds.

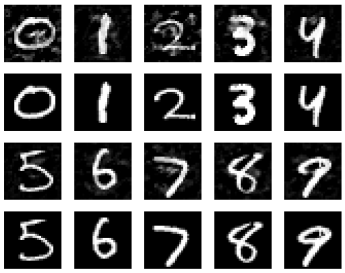

Figure 3: Adversarial examples crafted for the MNIST dataset against a classifier fortified with RFN. They are misclassified for more than 50 times in 100 parallel runs.

encoding is applied to each pixel of the input before feeding it to the classifier to discretize it. The way it works is as follows: for each pixel p of the image the k-level thermometer-encoding $\tau(p)$ is a k-dimensional vector where

$$\tau(p)_j = \begin{cases} 1, & \text{if } p \geq j/k \\ 0, & \text{otherwise} \end{cases}$$

and $\tau(p)_j$ is the j-th element of the vector. For example, for a 10-level thermometer encoding, $\tau(0.33) = 1110000000$. Since this function does a discrete transformation, it is not possible to back-propagate gradients through it. Therefore, an adversary can't craft adversarial examples for it using traditional white-box techniques.

4.2.1 Our Evaluation. For evaluating the effectiveness of SST against this defense, we used the model trained by Athalye et al. in [1], which is a wide ResNet model [38] fortified by thermometer encoding trained on CIFAR-10. For training this model, the adversarial training technique introduced by Madry et al. in [20] is also used for more robustness.

For attacking this model, we trained multiple substitute models with different levels of random noise. The model architecture we used as our substitute model is described in Table 7 in the Appendix. We trained four copies of this model on the CIFAR-10 test set, which we refer to as A, B, C and D. More specifically, we created a new dataset by replicating the CIFAR-10 test set eight times and adding different levels of random noises to it. We trained each substitute model with the training procedure we described in Section 3. The range of noises we added for training model A was $[-\frac{2}{255} \times i, \frac{2}{255} \times i]$ for $i \in [1, 8]$, where i was incremented for each replica. This range for Model B, C and D was $[-\frac{3}{255} \times i, \frac{3}{255} \times i]$, $[-\frac{4}{255} \times i, \frac{4}{255} \times i]$, and $[-\frac{5}{255} \times i, \frac{5}{255} \times i]$ respectively. Each of the substitute models was trained by the Adam optimizer as follows: 6 epochs with lr=0.001, 3 epochs with lr=0.0005, 3 epochs with lr=0.0001, 3 epochs with lr=0.00005, 3 epochs with lr=0.00001, 3 epochs with lr=0.000005, and 3 epochs with lr=0.000001.

Substitute Model(s)	Success Rate	L2 Norm	Time
A,B,C,D	100%	2.79	69 sec
A,B	99%	3.14	58 sec
C,D	99%	2.96	56 sec
A	96%	3.46	51 sec
B	99%	3.52	51 sec
C	97%	3.45	51 sec
D	99%	3.54	51 sec

Table 2: Results of applying our attack against thermometer encoding.

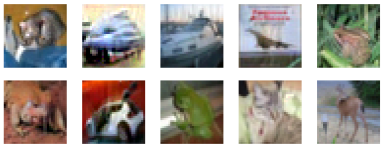

Figure 4: Adversarial examples crafted for the CIFAR-10 dataset against a classifier fortified with thermometer encoding using 4 substitute models.

We finally crafted adversarial examples using these models. For the first 100 images in the CIFAR-10 test set that were predicted correctly by the robust model, we set the total run to 3 and total iterations to 600. After every 100 iterations, we restarted the perturbation randomly to reduce the impact of sticking in a local minimum. Table 2 shows the success rate and average l2 norm for different scenarios in addition to average time for crafting one adversarial example. Figure 4 shows the adversarial examples generated using all 4 models. The reason that we couldn't find an adversarial example in some cases is that our substitute models didn't approximate the decision boundaries of the target model well enough in those cases. After some iterations, because of the values we chose for κ, $Z^{sub}(x')_t$ becomes greater than $max_{i \neq t}(Z^{sub}(x')_i) + \kappa$. As a result, the loss function becomes a constant value, with a gradient 0. Thus, newer perturbations won't be added to the current perturbation and the attack doesn't progress. We speculate that this problem can be solved by choosing a higher value for κ for those cases where the attack fails. The cost is a higher level of perturbation.

5 EVALUATION OF SST AGAINST DETECTING DEFENSES

In this section we evaluate the effectiveness of SST against the SafetyNet [19] and Defense-GAN [29] detecting defenses.

5.1 SafetyNet

Metzen et al. in [22] introduced a way to detect adversarial examples by augmenting the classifier with another DNN which acts as a detector. This detector network is trained with the outputs of some intermediate layer of the original classifier while adversarial examples and legitimate examples are fed into it. Later, Carlini et al. in [5] showed that an adversary can craft adversarial examples against this defense by changing the loss function such that an adversarial example can be crafted by back-propagating through both the original classifier and detector.

But, Lu et al. in [19] introduced another mechanism for adding a detector called SafetyNet. In SafetyNet, the detector is still connected to the output of some late layer of the classifier. But they used two techniques which make it impossible for the adversary to get any gradient from the detector. The first technique is quantization, in which the outputs of ReLU is quantized at some specific thresholds. The other technique is using a support vector machine (SVM) with a radial basis function (RBF) kernel as the detector. This provides no useful gradient to the adversary. They also observed that "there is a trade-off between classification confidence and detection easiness for adversarial examples. Adversarial examples with high confidence in wrong classification labels tend to have more abnormal activation patterns, so they are easier to be detected by detectors." As a result, the classification confidence is also considered in SafetyNet. For each input, the ratio of the second highest classification confidence to the highest classification confidence is calculated, and if it is bigger than a specific threshold that example is rejected. For our experiments, we set this threshold to 0.25, as suggested in the original paper.

5.1.1 Our Evaluation. Since the code for SafetyNet was not published, we implemented their defense ourselves on a model trained on MNIST. The model architecture we used for training is described in Table 8 in the Appendix. We trained this model with the Adam optimizer: 3 epochs with lr=0.001 and 3 epochs with lr=0.0001. The accuracy of this model on the MNIST test set was 99.15%, and the average confidence of correctly classified images was 99.63%.

To train the detector, we first generated non-targeted adversarial examples for the first 5000 samples of the training set using the C&W attack. For training the detector, we used the outputs of the first fully connected layer (layer 4), and we quantized them into four bins before feeding them to the SVM with the RBF kernel. Since the SafetyNet paper does not describe how the thresholds for quantization should be chosen, we chose them as follows: We fed our training data to the classifier and collected the outputs of layer 4. We sorted all the positive values from this data and found the 1st quartile Q_1, median, and 3rd quartile Q_3 and used them as thresholds for quantization. So all of the 0s, and any value less than the 1st quartile, were converted to the middle of that bin (i.e. $\frac{(Q_1-0)}{2}$). Any value between the 1st quartile and median was converted to the middle of the second bin (*i.e.,* $Q_1 + \frac{median-Q1}{2}$), and so on. Note that our attack works regardless of the way quantization thresholds are chosen. We decided to choose thresholds in this way as it gave us a good accuracy for training the detector. After training the detector in this way, it could achieve 95.15% accuracy on the MNIST test set.

In order to attack this defense, we trained two models with the logits of the classifier in the same way we did for RFN. We also trained another model with probability scores of the detector while the same data set was fed into the classifier. The architecture of the substitute classifier was same as the one we used for attacking RFN. The architecture of the substitute detector can be found in Table 9 in the Appendix. Note that we assumed that the attacker doesn't know where the detector is connected or what the input to it is. So, the substitute detector is trained on raw pixel values.

With the procedure we introduced in Algorithm 1, we crafted adversarial examples against SafetyNet. We set the total run value to 3 and total iterations to 300. The only difference was that in the inner loop we checked three things to make sure that the defense is bypassed. First, we checked to make sure that the adversarial example fools the classifier. Second, we checked that the confidence ratio is less than 0.25, and finally we checked to make sure the detector predicts it as a legitimate sample. We crafted adversarial examples for the first 100 samples that are classified correctly and predicted as legitimate samples by the detector. The average l2 norm of crafted adversarial examples was 3.37, and the success rate was 98%. You can see a few samples in Figure 5. In this figure the adversarial examples are in the first and third rows and corresponding legitimate examples are in the second and forth rows.

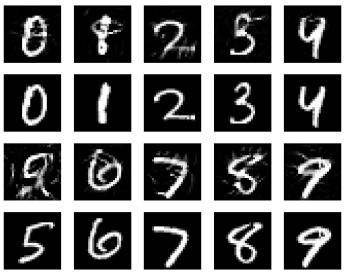

Figure 5: Adversarial examples crafted against SafetyNet for the MNIST dataset.

5.2 Defense-GAN

Samangouei et al. in [29] introduced a defense that makes a classifier more robust against adversarial examples. They also provided a mechanism to detect adversarial examples in case an attacker could fool the classifier. They called this defense, Defense-GAN as they used a Generative Adversarial Network (GAN) as part of their defense. A GAN consists of a generator G and a discriminator D in which, given a random vector z the generator tries to generate samples from a particular distribution and the discriminator tries to discriminate between a real sample and a generated sample by G. After training a GAN, the generator learns to generate a sample from that distribution. It is also shown that given an image x from that distribution, a vector z can be found such that the reconstruction error $||G(z)-x||_2 \longrightarrow 0$. So, this vector can be found by choosing a random vector z and then minimizing $||G(z)-x||_2$ with gradient descent for L steps and updating z in each step. In their defense, a GAN is first trained on legitimate examples.

Then at test time, given an image x and a random vector z, they first project x into the range of the generator using L steps of GD to minimize $||G(z)-x||_2^2$, and they do it R times for R different random vector $z_1, z_2, ...z_R$. Then, they choose the vector z^* that has a smaller reconstruction error and feed $G(z^*)$ to the classifier. They argued that by using this technique, a lot of adversarial perturbations will be removed from an image before feeding it to the classifier. Also, intuitively, given an image x after L iterations of GD, the reconstruction error is small if x was a legitimate example and large if it is otherwise. They used this property as their detection mechanism. So, given a threshold θ if $||G(z^*)-x||_2^2$ is larger than θ it is considered as an adversarial example and otherwise legitimate.

5.2.1 Our Evaluation. In order to evaluate the effectiveness of SST against Defense-GAN, we first trained a GAN with their published

code for 200,000 iterations. During the test time, we set $L = 200$ and $R = 10$.

The model architecture and parameters we used for the GAN's generator and discriminator are the same as those that were used in [29]. For the classifier, we used the model described in Table 11 in the Appendix. After training, the accuracy of this model on the MNIST test set was 99.26%. The accuracy on the output of the generator was 97.13%. Since it is not specified in the paper how θ should be chosen, we chose it in a way such that all correctly classified legitimate images are considered legitimate. Based on 9713 samples that were classified correctly, we set $\theta = 0.0536$, which is the largest reconstruction error for a correctly classified sample.

The architecture we used for the substitute model is the same as the one we used for attacking RFN. The dataset we used for training the substitute model was 7 replications of the MNIST test set, where a specific amount of random noise was added for each replication. The range of random noise we added was $[-0.15 \times i, 0.15 \times i]$ for $i \in [1, 7]$, where i was incremented for each replica. We trained four substitute models with this dataset. Crafting adversarial examples with this approach is very slow because querying Defense-GAN takes a long time. This makes training substitute models very slow, and in the process of crafting adversarial examples, we have to query Defense-GAN again in each iteration. We decided to craft adversarial examples with this approach for 15 samples. The average time was 156 seconds and success rate was 80%, and the average l2 norm of successful adversarial examples was 4.00. We speculate that our attack is less successful against this defense because of the generator. The generator removes a lot of random noise from our augmented dataset before feeding it to the classifier. In other words, it maps several distant inputs in our dataset to very close points before feeding them to the classifier. This behavior results in a smaller amount of variation in logits and makes SST less effective. One way to reduce this impact is to query the target model for a larger dataset, but this would result in a process that is very slow.

In our second attack, we trained a substitute model on the MNIST test set plus random noise with a range $[-0.95, 0.95]$ for 20 epochs, and in each epoch we updated the random noise. We didn't query Defense-GAN for training this model and used the default class labels and cross entropy loss for training. In order to make the crafting process faster in each iteration, we first checked whether the current adversarial example can fool a model augmented by Defense-GAN with $L = 30$ and $R = 1$. If we could fool it, we then checked it against the model with default parameters ($L = 200$ and $R = 10$). Since Defense-GAN has stochastic behavior for the same input in different runs, the output might be different. Thus, it is not clear when we should consider an attack successful. So, we considered an attack successful if the crafted adversarial example could fool the Defense-GAN in three consecutive runs. The sample adversarial examples crafted against this defense can be found in Figure 6. The samples are the result of running the attack while setting the target class to the second, third, and forth most probable classes, scored by the original model and choosing the least perturbed one. The success rate for the first 100 samples in the test set, which were classified correctly by the robust classifier, was 100%, and the average l2 norm of perturbations was 3.23. The maximum reconstruction error was 0.0431, which is less than θ.

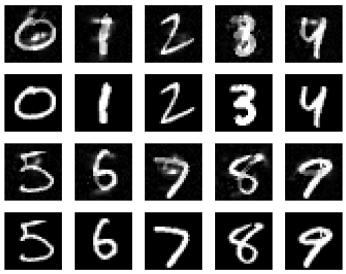

Figure 6: Adversarial examples crafted against Defense-GAN for the MNIST dataset.

Therefore, all of the crafted adversarial examples are considered as legitimate examples by the detector. The average reconstruction error was 0.0157. Note that this attack is even more powerful than the first approach, as the attacker doesn't know anything about the attack, the detection mechanism, or the classifier's parameters.

6 DISCUSSION OF RELATED WORKS

In this section, we evaluate the aforementioned defenses with two types of black-box attacks (mentioned in Section 2.3) that can also be used to craft adversarial examples with no knowledge about a defense and no need to be tailored towards different defenses. While black-box attacks might be more practical than the gray-box setting we considered, as our experiments in this section show, they are not good for evaluating new defenses, such as those we considered in this paper, as in many cases they can't find an adversarial example or the required perturbation is so high that it makes it hard even for a human to label correctly.

6.1 Black-Box Attack with Jacobian based Dataset Augmentation

Papernot et al. in [26] introduced a black-box attack that many researchers have used to show the robustness of their defense in a black-box setting. Similar to our attack, the authors in [26] showed how to attack a model using a small synthetic dataset without having access to the DNN's parameters or knowing about the defense that is in place. Their attack was of the non-targeted attack type in that they only made the model to mis-classify the inputs. The threat model they considered was different from what we considered in this paper. They assumed that an attacker only can send input to the target model and observe its predicted class, where we assume the the attacker also has access to the logits.

To attack the target model, they trained a substitute model to approximate the target model decision boundaries through a procedure called Jacobian-based Dataset Augmentation. The way it works is as follows: First they collect an initial small dataset. Then, they label this dataset using an Oracle (black box) model. Next, they train a substitute model using this dataset. Then, for each input x in their dataset, they evaluate the sign of the Jacobian matrix dimension corresponding to the label assigned to x by the oracle: $sgn(J_F(x)[O(x)])$, where F is the substitute model and $O(x)$ is the label assigned to the input by oracle. They then augment their initial dataset with new points created as follows: $x_{new} = x + \lambda.sgn(J_F(x)[O(x)])$, where λ is a hyper parameter. Finally, they repeat these steps for a few

iterations (substitute training epochs) so the substitute model can approximate the oracle decision boundaries.

After training a substitute model, the attacker crafts adversarial examples for the substitute model with the help of the JSMA and FGSM approaches in order to transfer them to the black-box model. We implemented this attack against the four defenses we considered, and the results can be found in Table 3. In all the cases, we crafted adversarial examples with the FGSM attack for the substitute model and checked what percentage of them can fool the robust model.

For evaluating all defenses, we trained the substitute model for 6 substitute training epochs using CleverHans library [25] and set the initial dataset to be the first 150 samples in the MNIST test set for RFN, SafetyNet, and Defense-GAN and first 150 samples in the CIFAR-10 test set for thermometer encoding. We also set $\lambda = 0.1$. The success rate for the adversarial examples crafted against detecting defenses (Defense-GAN and SafetyNet) shows the ones that could fool the classifier and are not detected by the detector. For example, for Defense-GAN, when $\epsilon = 0.5$, 92 out of 100 samples could fool the classifier, but all of them were detected as adversarial examples by the detector. The substitute model we used for attacking Defense-GAN was Model 1 described in Table 11 in the Appendix. The substitute model used for attacking SafetyNet was the same as the one we used in our attack. We evaluated RFN by setting $\epsilon = 0.25$. This is the same value we used for evaluating our attack against RFN in 3 different scenarios, which are referred to as RFN-50, RFN-70 and RFN-90. For example, for RFN-50, 47 samples could fool the classifier more than 50 times in 100 parallel runs. For evaluating this attack against thermometer encoding, we used the same model as the one we used in our attack for the substitute model. Figure 7 shows the generated adversarial examples against this defense at different value of ϵ. The rows are for $\epsilon = \frac{8}{255}$, $\epsilon = \frac{16}{255}$, $\epsilon = \frac{24}{255}$, $\epsilon = \frac{32}{255}$, and $\epsilon = \frac{64}{255}$ respectively. As can be seen, it becomes hard even for human eyes to classify these images correctly after $\epsilon = \frac{32}{255}$. Note that for RFN with the same level of distortion, the success rate of our approach is 2 times, 2.78 times and 3.4 times better than this black-box attack for RFN-50, RFN-70 and RFN-90 respectively. For thermometer encoding, with our approach the adversary could reach 100% success rate, but when using this black-box attack, even when $\epsilon = \frac{64}{255}$ and the images are unrecognizable by human eyes, the adversary could only reach 69% success rate. For detecting defenses, our approach also shows superiority and its success rate is at least 10 times better than this attack.

Defense Type	Success Rate	L2 norm	eps
RFN-50	47%	5.15	0.25
RFN-70	33%	5.12	0.25
RFN-90	20%	5.11	0.25
Thermometer	1%	1.73	$\frac{8}{255}$
Thermometer	7%	3.40	$\frac{16}{255}$
Thermometer	14%	5.12	$\frac{24}{255}$
Thermometer	32%	6.83	$\frac{32}{255}$
Thermometer	69%	13.12	$\frac{64}{255}$
SafetyNet	0%	NA	0.2
SafetyNet	0%	NA	0.3
SafetyNet	0%	NA	0.4
SafetyNet	0%	NA	0.5
Defense-GAN	7%	6.25	0.3
Defense-GAN	0%	NA	0.5

Table 3: Success rate of Jacobian-based Data Augmentation attack against different defenses. RFN, SafetyNet, and Defense-GAN were evaluated on MNIST. Thermometer encoding was evaluated on CIFAR-10.

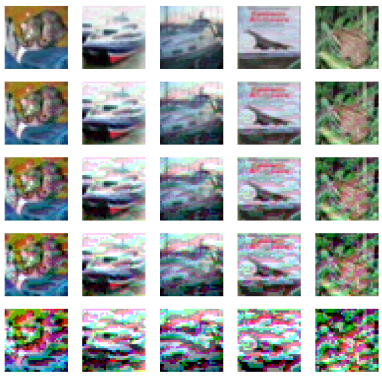

Figure 7: Adversarial examples crafted for CIFAR-10 to fool a classifier fortified with thermometer encoding at different levels of perturbation.

6.2 Transferable Adversarial Examples

Liu et al. in [18] designed a different targeted black box attack. In contrast to our attack, they considered a different threat model where they don't need to query the target model to get the outputs for different inputs, and they don't need to train a substitute model. Instead of training a substitute model, they leverage an ensemble of pre-trained models to craft adversarial examples that can be transferred to another model with their targeted labels.

The authors showed that by generating adversarial examples for an ensemble of pre-trained neural networks, it is more likely to transfer them to another classifier. Formally, given k white-box models with softmax outputs being $F_1, ..., F_k$, they solve the following optimization problem:

$$argmin_\delta - \log((\sum_{i=1}^{k} \alpha_i F_i(x')).1_{y_{target}}) + \lambda d(x, x')$$

where $\sum_{i=1}^{k} \alpha_i F_i(x^*)$ is the ensemble model, α_i are the ensemble weights, and $\sum_{i=1}^{k} \alpha_i = 1$. $d(x, x')$ is the distance function between the original image and the perturbed image which can be l_2 norm. λ is a hyper parameter which controls the amount of distortion and the success rate. Increasing λ results in a larger distortion, while also increasing the success rate (likelihood of being able to fool the target classifier). Solving this optimization problem basically means that the attacker wants to maximize the score of the targeted label in all of the classifiers while keeping the amount of distortion small.

We implemented this attack against the four defenses we considered, and the results can be found in Table 4. For attacking RFN, SafetyNet, and DefenseGAN, we trained four substitute models described in Tables 11, 12, 13, and 14 on the MNIST training set for

Figure 8: Adversarial examples crafted with different Îż for the MNIST dataset.

10 epochs each and with Adam optimizer (lr=0.001). The architecture of these models can be found in the Appendix. For generating adversarial examples, as in the original paper, we used Adam with lr=0.001 to optimize the above objective and $\alpha_i = 0.25$. For generating each adversarial example, we did 300 iterations of GD. For SafetyNet, we crafted adversarial examples by changing the λ parameters. As you can see in Table 4, when $\lambda = 0.001$ the success rate is only 17%. However, even in this case, the amount of perturbation is too high. Samples of adversarial examples generated by this method can be found in Figure 8. For the first row λ is 0.1, for the second one λ is 0.01, and for the last one λ is 0.001.

For applying this attack on thermometer encoding, since our robust model was trained on CIFAR-10, we trained four other models to generate adversarial examples with them, and we trained all of them with the CIFAR-10 training set. The models we trained for this purpose were VGG-19 [32], wide ResNet [38], ResNet-50 [11] and NIN [17]. After training the VGG model reached to 93.41% accuracy, the wide ResNet model reached to 92% accuracy, the NIN model reached 90.29% accuracy, and the ResNet-50 reached 94.06% accuracy. We trained all of these models with 4 Tesla K80 GPUs. The training time for these models were 2, 7, 1, and 6 hours respectively. This is a much longer training time than in our approach, in which we trained a smaller model, and it only took a few minutes. The robust model's accuracy was also 88.59%. For the attack we used the same hyper parameters as the ones we used against three other defenses. Generating each adversarial example took 63 seconds on average. Samples of crafted adversarial examples, with different λ, are shown in Figure 9. Note that for RFN, in an unbounded attack, our approach could reach 100% success rate. Using this attack, amongst our experiments, the best case success rate was 93% (For RFN-50 when $\lambda = 0.001$). The average l2 norm of adversarial examples crafted by our approach against RFN-50 was 2.13, which is almost 3 times smaller than the average l2 norm of crafted adversarial example using this approach. For thermometer encoding, in the best case, using this attack it could reach 39% success rate. Using our approach, we could reach 100% success rate, and the average l2 norm of adversarial examples found by our approach in the best case (when we used 4 substitute models) is 2.68 times smaller than the average l2 norm of crafted adversarial examples found by this approach. For detecting defenses, our approach is also more successful and at least 5 times better than this attack.

7 CONCLUSION

In this paper we described a way to craft adversarial examples against deep neural network models that leverage mechanisms to protect themselves against adversarial examples. We evaluated our approach against fortifying and detecting defenses. We showed that

Defense Type	Success Rate	L2 norm	λ
RFN-50	26%	1.74	0.1
RFN-70	16%	1.74	0.1
RFN-90	1%	1.59	0.1
RFN-50	84%	3.52	0.01
RFN-70	54%	3.44	0.01
RFN-90	30%	3.27	0.01
RFN-50	93%	6.31	0.001
RFN-70	88%	6.23	0.001
RFN-90	81%	6.15	0.001
Thermometer	4%	2.01	0.1
Thermometer	25%	5.11	0.01
Thermometer	39%	7.47	0.001
SafetyNet	0%	NA	0.1
SafetyNet	2%	3.83	0.01
SafetyNet	17%	5.58	0.001
SafetyNet	17%	6.26	0.0001
Defense-GAN	1%	1.62	0.1
Defense-GAN	10%	2.78	0.01
Defense-GAN	18%	6.33	0.001

Table 4: The success rate and average l2 norm of crafted adversarial examples by Liu et al. work against different defenses. RFN, SafetyNet, and Defense-GAN were evaluated on MNIST. Thermometer encoding was evaluated on CIFAR-10.

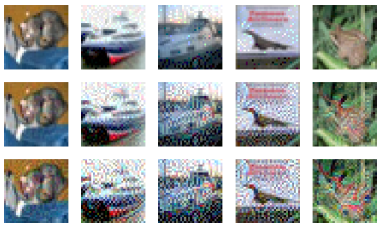

Figure 9: Adversarial examples crafted with different Îż for the CIFAR-10 dataset.

an adversary can craft adversarial examples without any knowledge about the type of defense used, defense parameters, model parameters, or training data. The adversary only needs to query the robust model and train one or more substitute models. We also evaluated two black-box attacks against the aforementioned defenses, but they performed poorly in comparison to our presented attack. We suggest that other researchers use our approach for benchmarking in cases where a defense prevents the attacker from calculating useful gradients from the target model.

ACKNOWLEDGMENT

This research was supported in part by the National Science Foundation under grants 1406192 (SaTC), 1652698 (CAREER), and 1700527 (SDI-CSCS). It also utilized the RMACC Summit supercomputer, which is supported by the National Science Foundation (awards ACI-1532235 and ACI-1532236), the University of Colorado Boulder, and Colorado State University. We would also like to thank the reviewers, and Nicolas Papernot in particular, for their valuable feedback and guidance.

REFERENCES

[1] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*. https://arxiv.org/abs/1802.00420

[2] Shumeet Baluja and Ian Fischer. 2018. Learning to Attack: Adversarial Transformation Networks. In *Proceedings of AAAI-2018*. http://www.esprockets.com/papers/aaai2018.pdf

[3] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks against Machine Learning at Test Time. In *ECML/PKDD*.

[4] 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/pdf?id=S18Su--CW

[5] 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*. ACM, 3–14.

[6] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy*. 39–57.

[7] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. DeepDriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2722–2730.

[8] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust Physical-World Attacks on Deep Learning Visual Classification. In *Computer Vision and Pattern Recognition*. IEEE.

[9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *The IEEE International Conference on Computer Vision (ICCV)* (2015), 1026–1034.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 770–778.

[12] Konstantinos Kamnitsas, Enzo Ferrante, Sarah Parisot, Christian Ledig, Aditya Nori, Daniel Criminisi, Antonio Rueckert, and Ben Glocke. 2016. DeepMedic for Brain Tumor Segmentation. In *Proceedings MICCAI-BRATS Workshop*. 18–22.

[13] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

[14] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

[15] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial Examples in the Physical World. In *International Conference on Learning Representations*.

[16] Yann LeCun, Corinna Cortes, and Christopher JC Burges. 1998. The MNIST database of handwritten digits. (1998).

[17] Min Lin, Qiang Chen, and Shuicheng Yan. 2014. Network In Network. In *International Conference on Learning Representations*.

[18] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2017. Delving into Transferable Adversarial Examples and Black-box Attacks. In *International Conference on Learning Representations*.

[19] Jiajun Lu, Theerasit Issaranon, and David A. Forsyth. 2017. SafetyNet: Detecting and Rejecting Adversarial Examples Robustly. In *The IEEE International Conference on Computer Vision (ICCV)*.

[20] 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*.

[21] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 50–56.

[22] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. 2017. On Detecting Adversarial Perturbations. In *International Conference on Learning Representations*.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan umaran, Daan ierstra, Shane egg, and Demis assabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (2015), 529–533.

[24] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2574–2582.

[25] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. 2018. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv preprint arXiv:1610.00768* (2018).

[26] 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

[27] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 372–387.

[28] Nicolas Papernot, Patrick 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 (SP)*. 582âĂŞ597.

[29] 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://arxiv.org/abs/1805.06605

[30] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. Human-level control through deep reinforcement learning. *International Conference on Learning Representations* (2016).

[31] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529 (2016), 484–489.

[32] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

[33] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[34] Tuan A. Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. 2016. Deep learning approach for Network Intrusion Detection in Software Defined Networking. In *International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 258–263.

[35] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2018. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*.

[36] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G., Xinyu Xing, C. Lee, and Xue Liu. 2017. Adversary Resistant Deep Neural Networks with an Application to Malware Detection. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1145–1153.

[37] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: Deep learning in android malware detection. In *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, Chicago, Illinois, USA, 371–372.

[38] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide Residual Networks. *arXiv preprint arXiv:1605.07146* (2016).

APPENDIX

The model architectures and parameters we used throughout the paper.

DNN Structure	784-784-784-784-10
Activation	Relu
Optimizer	SGD
Learning Rate	0.1
Dropout Rate	0.25
Batch Size	100
Epoch	25

Table 5: The model architecture and hyper parameters used for training the model for RFN evaluation.

Layer Type	Parameters
Convolution + ReLU	$64, 3 \times 3, 1$
Convolution + ReLU	$64, 3 \times 3, 1$
Convolution + ReLU	$64, 3 \times 3, 1$
Max Pooling	$2 \times 2, 2$
Convolution + ReLU	$64, 3 \times 3, 1$
Fully Connected + ReLU	2048
Fully Connected	10
Softmax	-

Table 6: The substitute model architecture and hyper parameters used for attacking RFN.

Layer Type	Parameters
Convolution + ReLU	$64, 3 \times 3, 1$
Convolution + ReLU	$64, 3 \times 3, 1$
Max Pooling	$2 \times 2, 2$
Convolution + ReLU	$128, 3 \times 3, 1$
Convolution + ReLU	$64, 3 \times 3, 1$
Max Pooling	$2 \times 2, 2$
Convolution + ReLU	$64, 3 \times 3, 1$
Fully Connected + ReLU	4096
Fully Connected + ReLU	1024
Fully Connected	10
Softmax	-

Table 7: The substitute model architecture and hyper parameters used for attacking Thermometer Encoding

Layer#	Layer Type	Parameters
1	Convolution + ReLU	$64, 3 \times 3, 1$
2	Max Pooling	$2 \times 2, 2$
3	Convolution + ReLU	$64, 3 \times 3, 1$
4	Fully Connected + ReLU	2048
5	Fully Connected	10
6	Softmax	-

Table 8: The model architecture used for evaluating SafetyNet.

Layer#	Layer Type	Parameters
1	Convolution + ReLU	$64, 3 \times 3, 1$
2	Convolution + ReLU	$64, 3 \times 3, 1$
3	Convolution + ReLU	$64, 3 \times 3, 1$
4	Max Pooling	$2 \times 2, 2$
5	Convolution + ReLU	$64, 3 \times 3, 1$
6	Fully Connected + ReLU	1024
7	Fully Connected + ReLU	512
8	Fully Connected + ReLU	512
9	Fully Connected	2
10	Softmax	-

Table 9: The substitute detector architecture used for attacking SafetyNet.

Layer Type	Parameters
Convolution + ReLU	$64, 5 \times 5, 1$
Convolution + ReLU	$64, 5 \times 5, 2$
Dropout	0.25
Fully Connected + ReLU	128
Dropout	0.5
Fully Connected	10
Softmax	-

Table 10: The model architecture used for evaluating Defense-GAN.

Layer Type	Parameters
Dropout	0.2
Convolution + ReLU	$64, 8 \times 8, 2$
Convolution + ReLU	$128, 6 \times 6, 2$
Convolution + ReLU	$128, 5 \times 5, 1$
Dropout	0.5
Fully Connected	10
Softmax	-

Table 11: The Model 1 architecture.

Layer Type	Parameters
Convolution + ReLU	$128, 3 \times 3, 1$
Convolution + ReLU	$64, 3 \times 3, 2$
Convolution + ReLU	$128, 5 \times 5, 1$
Dropout	0.25
Fully Connected + ReLU	128
Dropout	0.5
Fully Connected	10
Softmax	-

Table 12: The Model 2 architecture.

Layer Type	Parameters
Fully Connected + ReLU	200
Dropout	0.5
Fully Connected + ReLU	200
Dropout	0.5
Fully Connected	10
Softmax	-

Table 13: The Model 3 architecture.

Layer Type	Parameters
Fully Connected + ReLU	200
Fully Connected + ReLU	200
Fully Connected	10
Softmax	-

Table 14: The Model 4 architecture.