Stochastic Substitute Training: A General Approach to Craft Adversarial Examples against Defenses which Obfuscate Gradients

Anonymous Author(s)

ABSTRACT

It has been shown that neural networks are vulnerable to adversarial examples. These examples are inputs similar to legitimate examples but are constructed in a way such that they are incorrectly classified or classified as the adversary's desired class. An adversary can craft adversarial examples 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 to prevent an adversary from crafting adversarial examples even in a white-box setting. There is also another line of research which has tried to detect adversarial examples instead of making a model more robust. Some of these defenses use the same class of technique in the detector which prevents the extraction of useful gradients to fool the detector. In this paper, we demonstrate how to craft adversarial examples for models which leverage these types of defenses. For those defenses which try to robust a model we show that an adversary can craft adversarial examples for them with no knowledge of the defense and for the other type with a very limited knowledge about the defense. We demonstrate our technique by applying it against two defenses which make models more robust and two defenses which detect adversarial examples.

1 INTRODUCTION

Deep learning has evolved in many areas. In applications such as image classification, deep neural networks can surpass human level performance [9]. These deep neural networks show promising results in tasks such as malware detection [32], autonomous driving [6], network intrusion detection [29], etc. They are also used for diagnosis in medical images. Kamnitsas et. al showed in [11] how they used a deep neural network with residual connection for brain tumor segmentation. 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 [20], defeating best human players in the game of Go [26], making a humanoid robot run [25], and resource management in a cluster with different resource types [18].

Despite their success in a wide range of applications, deep neural networks, like other traditional classifiers, suffer from a problem known as vulnerability to adversarial examples. When working with images, an adversarial example is an image which is carefully and slightly modified to make a classifier predict it incorrectly and potentially as the attacker's desired class. The perturbation that is added to these images are imperceptible in many cases to a human observer. Therefore, a human is likely to classify the images as they did before the alterations. It is also shown that this attack is applicable in the physical world. Eykholt et al. in [7] showed that an adversary can stick a few stickers to a stop sign to fool a classifier (possibly deployed on an autonomous car) to predict it as a speed limit sign.

Due to the threat that adversarial examples have, many researchers have tried to find a solution for them. These researches can be categorize into two different groups. In one line of work, they 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 [3, 31]. We refer to these defenses as "fortifying defenses". In another line of work, researchers have tried to distinguish between legitimate examples and adversarial examples using some detection mechanisms [16, 24]. We refer to these defenses as "detecting defenses" In order to craft an adversarial example, the attacker needs to calculate the gradients of loss function with respect to the input. Carlini et al. in [4] 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. These 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. Athalye et al. in [1] demonstrated three ways to craft adversarial examples against these defenses in which the attacker needs to know about the defenses, their parameters, and model parameters. In this paper, we take a look at these defenses and demonstrate a general way to break them without any knowledge about the model's parameters or defense parameters and without accessing the training dataset. 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 them. More specifically, 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 which obfuscate gradients such that an adversary cannot craft adversarial examples for them using old methods such as those introduced in [5, 8, 23, 28].
- As a case study for fortifying defenses We evaluate random feature nullification (RFN) [31] and thermometer encoding [3] on MNIST and CIFAR-10 datasets respectively. With our approach, 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.
- As a case study for detecting defenses we evaluate SafetyNet [16] and Defense-GAN [24] on the MNIST dataset. and we show how an adversary can bypass these detection methods.

- Finally, we evaluate two other black-box attacks that others have used to evaluate their designed defenses [15, 22] to benchmark against our approach.

The rest of this paper is organized as follows: in Section 2, we provide the reader with an overview and background information. Then, in Section 3, we introduce our approach for attacking. Next, in Section 4, we do a case study and evaluate our approach on fortifying defenses. Then, in section 5, we evaluate our approach on detecting defenses. In Section 6, we implement two other black-box attacks against the aforementioned defenses for benchmarking purpose, and finally in Section 7, we conclude the paper.

2 OVERVIEW

In this section we first briefly explain how deep neural networks work for image classification and then introduce the symbols we use in this paper. We then go over how an adversary can craft an adversarial example. Furthermore, we introduce the datasets that we used in our evaluations and talk about the threat model we consider in this paper.

2.1 Deep Neural Networks

A deep neural network (as a classifier) is a non-linear function which maps an input to a probability vector where each of its elements correspond to a class score. The one that has the larger score is considered as its prediction. A deep neural network consists of multiple layers and the layers are connected to each other sequentially such that the output of one layer becomes the input of the next layer. Here's my go at it: Each layer has a set of parameters which are initialized randomly. Each layer applies a non-linear transformation to its inputs and sends them to another space with a different dimension. By varying those parameters, the output of the classifier gets changed. The goal is to find those parameters such that for most of the inputs it predicts their labels correctly which means that the probability corresponded to their true label should be larger than others. Figure 1 illustrates a neural network and its different components. In this figure, the input is one of MNIST samples which is number 0. The parameters of the classifier are shown by θ which are initialized randomly and should be tuned during the training phase. The output is shown by F which is the probability vector: $\sum_{i=0}^{9} F_j = 1$. The last layer of this classifier is a softmax ($\sigma(.)$) function which converts its inputs to probabilities as follows:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e_k^z} \text{ for } k = 1, 2, ..., K$$

Where K is the total number of classes. Since the parameters are initialized randomly, the classifier would output random probabilities for different inputs before training. For example, this classifier "thinks" that the input is 8 with 60% probability and it is 0 by 1% probability. Since F_8 is greater than others, the input would be classified as 8 by this classifier. In this figure, you can also see the true label corresponded to image zero which is a one-hot vector with its first element as 1 and other elements are 0. These labels are used in training phase to tune the parameters of the network.

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 corresponded to the true label for each input. For example, if the input is number 7 we have to maximize

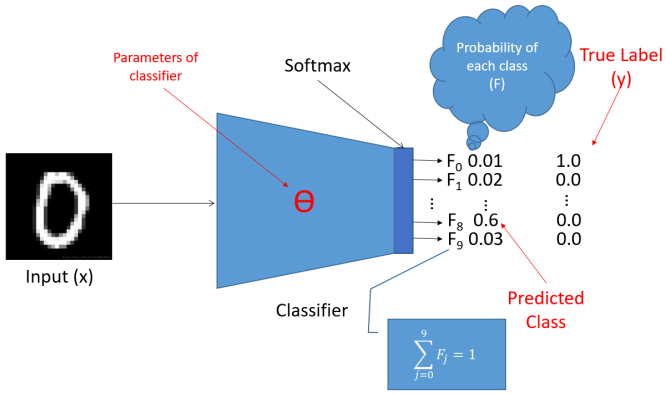

Figure 1: Illustration of a DNN classifier.

$F_7(x)$, and if it is number 5 we have to maximize $F_5(x)$ and so on. 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 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 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, we use a technique called gradient descent. Figure 2 illustrates this technique for a simple function $G(\theta) = \theta^2$. As you can see in this figure, at any point if we move the input θ in the opposite direction of the slope (derivate of function with respect to its input) the value of the function decreases. So in order to minimize the function $G(\theta)$ with respect to θ we can take a small step in the opposite direction of slope and then update the value of theta and do it iteratively until we reach to the minimum value as follows:

$$\theta = \theta - \alpha \frac{\partial G}{\partial \theta}$$

Where α shows the magnitude of each step and $\frac{\partial G}{\partial \theta}$ is corresponded to the slope and minus sign means that we want to move in the opposite direction of the slope.

This technique also works for the loss function of a DNN even though it is not a convex function. The only difference is that instead of one variable there are multiple parameters. So we need to take partial derivatives with respect to each of the parameters and update them separately.

2.2 Notations

In the rest of this paper we use these notations:

- x: the legitimate (clean) input. $x \in [0, 1]^m$, where m is the number of pixels in an image.

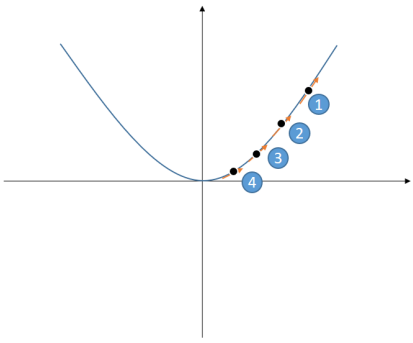

Figure 2: Illustration of gradient descent. The horizontal axis shows θ. The vertical axis shows $G(\theta) = \theta^2$. By moving in the opposite direction of the slope at each step we can minimize function G.

- 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

The process for crafting an 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. This process leads to a targeted attack. 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 oppose 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 [5] 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:

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

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 which determines the trade-off between the amount of distortion and the growth of the target score. As c grows the the amount of distortion and success probability grows. They showed that by using this function they can craft adversarial examples for 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 which 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 the κ the confidence score of 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 we refer the reader to these papers: [2, 8, 21, 23, 28].

Szegedy et al. in [28] showed that in many cases an adversarial example, which is built using one classifier, can fool another classifier which has different architecture and parameters. This property is called the transferability of adversarial examples. Later Tramèr et al. in [30] showed that augmenting training data with adversarial examples generated by a few fixed, pre-trained models improves the robustness of a model in the face of these types of transferable black-box attacks significantly.

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. Among the techniques which are used for these defenses are removing adversarial noise by transforming the input before feeding it to the classifier, quantization or discretization of input, and randomization of input or the model.
- Detection Defenses: For these types of defenses the classifier may predict an adversarial example wrongly but there is an adversarial detection mechanism which makes the whole model reject those cases. Among the techniques which are used for these defenses, we can refer to augmenting the classifier with another DNN (or any other model)

to classify the input as legitimate or adversarial. People also use some statistics which they assume are co-related with adversarial examples for detecting them.

2.5 Datasets

In this section we describe the datasets that we use for evaluation of our approach to attack aforementioned defenses.

2.5.1 CIFAR-10. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images [12]. The classes which are in this dataset are as follows: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. The classes are completely mutually exclusive. So, for example, there is no overlap between automobiles and trucks.

2.5.2 MNIST. In MNIST dataset each sample is an image of a handwritten number between 0 and 9 [13]. This dataset consists of 60000 examples in train set and 10000 examples in test set. The size of each image in this data set is 28 by 28 and they are gray-scale. Since the size of each image in this dataset is pretty small, crafting an adversarial example for the models which are trained on MNIST is harder than those which are trained on CIFAR-10. It is easier for human eyes to notice a slight change in small gray-scale images compared to those images which have more pixels. To have a better sense, consider that someone adds a liter of water to a cup which is half full. It is easier for an observer to notice this change compared to the case where someone adds a litter of water to a pool.

2.6 Threat Model

In this paper we consider two different threat models:

- 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 STOCHASTIC SUBSTITUTE TRAINING

In order to attack the robust classifiers with defenses that obfuscate gradients, we use the transferability property of adversarial examples. In general, we add a random noise to the set of images we want to craft adversarial examples for, specifically 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 3 illustrates this process.

For training this model, instead of using default cross entropy loss, we use mean square error between the substitute models logits and logits we got from robust model as our loss function.

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

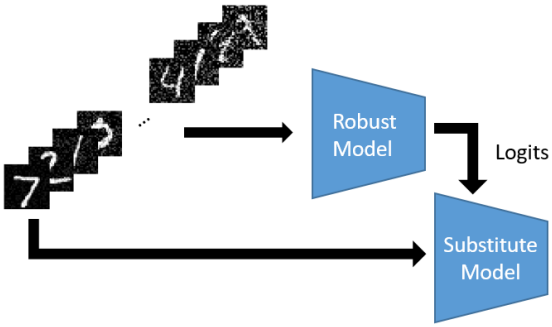

Figure 3: Training step for crafting adversarial examples with substitute model against robust classifiers

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's decision boundaries for that dataset very close to the robust model which makes transferability to robust model easier, and since the substitute model is differentiable, we can craft adversarial examples for it using some 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 noise 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 a different level of random noise. This is because if we add low level noise, the model might be very robust and that level may not be sufficient to fool the classifier. On the other hand, if we add high level noise, the model may not be that robust and unnecessary perturbation would be added to the adversarial example. The reason we use logits instead of class probabilities is that the effect of low level noise is more obvious in logits compared to probabilities. Using probabilities may not capture the impact of small random noise because of floating point precision. We empirically found that for complex datasets training multiple copies of a model with different random noise reduces the required 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 approximate 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 that we 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 current perturbation can fool the robust model. If so, we increase the c parameter in C&W loss function to craft adversarial examples with smaller amount of distortion in next iterations. We also keep decreasing the value of c in each iteration while we couldn't fool the robust model 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 the 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.

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 FORTIFYING DEFENSES

In this section we evaluate Random Feature Nullification (RFN) [31] and Thermometer Encoding [3] defenses with SST.

4.1 Random Feature Nullification

Wang et al. in [31] 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 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 uniform distribution is used for generating each row of I_p. During the 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 while adversary is allowed to change the value of each pixel by 0.25.

4.1.1 Our Evaluation. In order to evaluate RFN, since the authors didn't publish their code, we trained a model with the same architecture and parameters they used in their paper. More specifically, the parameters can be found in table 8 in appendix. As for the hyper parameters of RFN we set the 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 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 noise was in $[-0.05, 0.05]$, for the next 10000 samples the noise was in $[-0.1, 0.1]$, and so on and so forth. 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 1. 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 model on legitimate samples and average l2 norm which is required to fool this model in different scenarios. First, we consider an input to be classified correctly if in 100 parallel run 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 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 2. The success rate in all cases was 100%. Figure 4 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 [3] introduced another defense to robust model against adversarial examples called thermometer encoding. Thermometer encoding prevents an adversary from calculating gradients for crafting adversarial examples. Thermometer 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 image the k-level thermometer-encoding $\tau(p)$ is a k-dimensional vector where

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 1: The substitute model architecture and hyper parameters used for attacking RFN

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

Table 2: The model accuracy and average l2 norm of required perturbation across different threshold

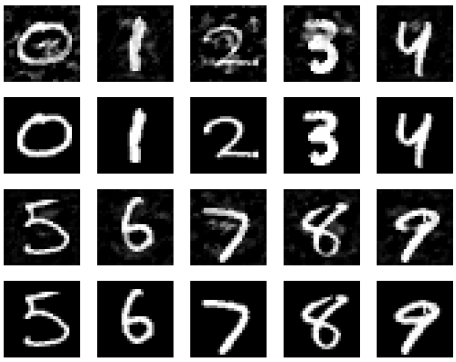

Figure 4: Adversarial examples crafted against a classifier fortified with RFN while they are misclassified for more than 50 times in 100 parallel run.

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

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. So, an adversary can't craft adversarial examples for it using old white-box techniques.

4.2.1 Our Evaluation. For evaluating this defense we used the model trained by Athalye et al. in [1] which is a wide ResNet model [33] fortified by thermometer encoding trained on CIFAR-10. For training this model, the adversarial training technique introduced by Madry et al. in [17] 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 3. We trained four

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

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 4: Results of applying our attack against thermometer encoding

copies of this model on the CIFAR-10 test set which we refer to them as A,B,C and D. More specifically, we created a new dataset by replicating CIFAR-10 dataset eight times and adding different level of random noise to it. We trained each substitute model with the training procedure we described in Section 3. Table 9 in appendix shows the range of random noise added to each dataset. Each of the substitute models was trained by 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.

We finally crafted adversarial examples using these models For the first 100 images in the CIFAR-10 test set which 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 local minimum. Table 4 shows the success rate and average l2 norm for different scenarios in addition to average time for crafting one adversarial example. Figure 5 also shows the adversarial examples generated using all 4 models.

5 EVALUATION OF DETECTING DEFENSES

In this section we evaluate SafetyNet [16] and Defense-GAN [24] with our attack.

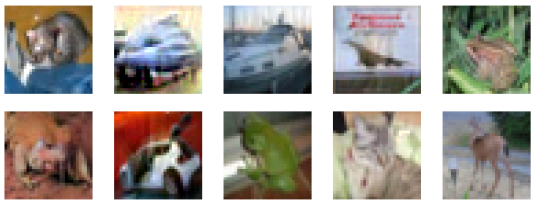

Figure 5: Adversarial examples crafted against a classifier fortified with thermometer encoding using 4 substitute models

5.1 SafetyNet

Metzen et al. in [19] introduced a way to detect adversarial examples by augmenting the classifier with another DNN which acts as a detector. This detector network was trained with the outputs of some intermediate layer of original classifier while adversarial examples and legitimate examples are fed into it. Later Carlini et al. in [4] 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 [16] introduced another mechanism for adding a detector which resists against this attack 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 makes 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 SVM with RBF kernel as detector which provides no useful gradient to the adversary to fool it. 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 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 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 10 in 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 MNIST test set was 99.15% and average confidence of correctly classified images was 99.63%. In order to train the detector, we first generated non-targeted adversarial examples for the first 5000 samples of train set using 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 to the SVM with RBF kernel. Since it is not described in the SafetyNet paper 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 and so forth. 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 MNIST test set.

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 5: The substitute detector architecture used for attacking SafetyNet

In order to attack this defense, we trained two models with the logits of the classifier 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 substitute detector can be found in table 5. 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 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 6. In this figure the adversarial examples are in first and third rows and corresponding legitimate examples are in the second and forth rows.

5.2 Defense-GAN

Samangouei et al. in [24] introduced a defense which makes a classifier more robust against adversarial examples. They also provided a mechanism to detect adversarial examples in case an attacker can fool the classifier. They called this defense, Defense-GAN as they used a Generative Adversarial Network (GAN) as part of their defense. GAN consists of a generator G and a discriminator D in

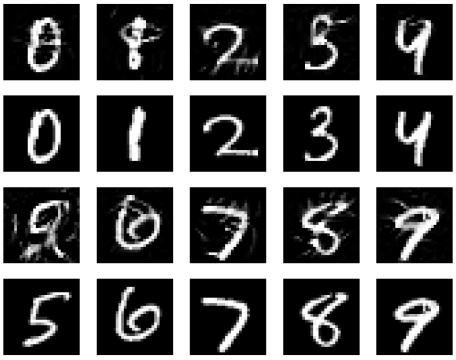

Figure 6: Adversarial examples crafted against SafetyNet.

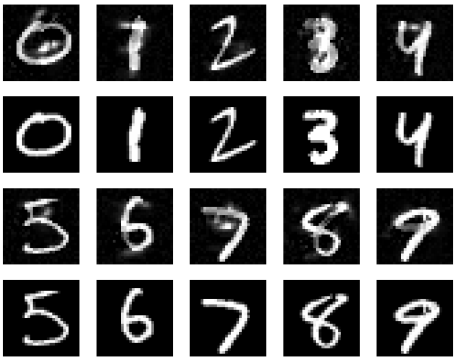

Figure 7: Adversarial examples crafted against defense-GAN.

which given a random vector z generator tries to generate a samples from a particular distribution and discriminator tries to discriminate between a real sample and a generated sample by G. After training a GAN, 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 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, lots of adversarial perturbation 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 otherwise it is large. 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 a adversarial example and otherwise a legitimate example.

5.2.1 Our Evaluation. In order to evaluate 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 same as those which were used in [24]. For the classifier, we used the model described in Table 12. After training the accuracy of this model on MNIST test set was 99.26%. The accuracy on the output of generator was 97.13%. Since it is not specified in the paper how θ should be chosen, we chose it in a way that all correctly classified legitimate images are considered legitimate. Based on 9713 samples who 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 models is the same as the one we used for attacking RFN. The dataset we used for training the substitute model was 7 replication of MNIST test set which a specific amount of random noise was added to 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 dataset. 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. Thus we introduce a faster approach for this defense.

In our second attack, we trained a substitute model on MNIST test set plus random noise with range $[-0.95, 0.95]$ for 20 epochs and in each epochs 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 crating process faster in each iteration, we first check whether the current adversarial example can fool a defense-GAN model with $L = 30$ and $R = 1$, and if we could fool it, then we check it against the model with default parameters ($L = 200$ and $R = 10$). Since, defense-GAN has stochastic behavior for same input in different runs the output might be different. Thus, it is not clear when we should consider an attack successful. So, we consider 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 7. 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 less perturbed one. The success rate for the first 100 samples in the test set, which were classified correctly by 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. 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 or the detection mechanism or the classifier's parameters.

6 DISCUSSION OF RELATED WORKS

In this section we evaluate the aforementioned defenses with two other black-box attacks.

6.1 Black-Box Attack with Jacobian based Dataset Augmentation

Papernot et al. in [22] introduced a black-box attack that many researchers used to show the robustness of their defense in a black-box setting. Similar to our attack, the authors in [22] showed how to attack a black-box model using a synthetic dataset without having access to DNN's parameters or knowing about the defense which is in place. Their attack was of the non-targeted attack type in which they only made the black box model to mis-classify the inputs. They assumed that an attacker only can send input to the target model and observe its predicted class. As opposed to our attack, the attacker doesn't have access to the class scores or logits.

In order to attack the black box 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 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 by 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 using the above approach, 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 6. In all the cases, we craft adversarial examples with 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 and set the initial dataset to be the first 150 samples in the MNIST test set and $\lambda = 0.1$. 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 12. The substitute model used for attacking SafetyNet was same as the one we used in our attack. We evaluated RFN by setting $\epsilon = 0.25$, the same value we used for evaluating our attack against RFN in 3 different scenarios which are referred by 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 run. For evaluating this attack against thermometer, we used the same model as the one we used in our attack for the substitute model. Figure 8 shows the generated adversarial examples against this defense at different value of ϵ. The rows are for $\epsilon = \frac{8}{255}$, $\epsilon = \frac{16}{255}$ and $\epsilon = \frac{24}{255}$, $\epsilon = \frac{32}{255}$, $\epsilon = \frac{64}{255}$ respectively. As you can see it becomes hard even for human eyes to classify these images correctly after $\epsilon = \frac{32}{255}$.

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 6: The model accuracy and average l2 norm of required perturbation across different threshold

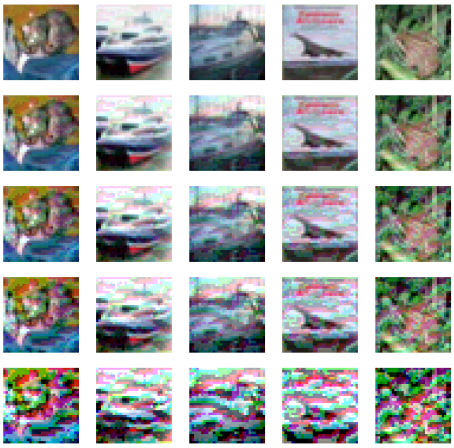

Figure 8: Adversarial examples crafted to fool a robust classifier with Thermometer Encoding at different level of perturbation.

6.2 Transferable Adversarial Examples

Liu et al. in [15] designed a different targeted black box attack. In spite of our attack, they don't need to query the target model to get the outputs of it on 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 to transfer 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, and α_i are the ensemble weights, $\sum_{i=1}^{k} \alpha_i = 1$. $d(x, x')$ is the distance function between original image and perturbed image which can be l_2 norm and a larger value of λ leads to a larger distortion and more success rate. Solving this optimization problem basically means that the attacker wants to maximize the score of targeted label in all 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 7. For attacking RFN, SafetyNet, and DefenseGAN, we trained four substitute models described in tables 12, 13,14, and 15 on the MNIST train and set each of them for 10 epochs and with Adam optimizer (lr=0.001). The architecture of these models can be found in the appendix. For generating adversarial examples, same as 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 the table, 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 9. For the first row $\lambda = 0.1$, for the second one $\lambda = 0.01$, and for the last one $\lambda = 0.001$

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 CIFAR-10 train set. The models we trained for this purpose were VGG-19 [27], wide ResNet [33], ResNet-50 [10] and NIN [14]. After training the VGG model reached to 93.41% accuracy, the wide ResNet model reached to 92% accuracy, the NIN model reached to 90.29% accuracy, and the ResNet-50 reached to 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 same hyper parameters as the ones we used against three other defenses. Generating each adversarial example took 63 seconds in average. Samples of crafted adversarial examples with different λ are shown in figure 10.

7 CONCLUSION

In this paper we described a way to craft adversarial examples against deep neural network models which leverage mechanisms to protect themselves against adversarial examples. We evaluated our approach against fortifying and detecting defenses. We showed that an adversary can craft adversarial examples without any knowledge about the type of defense used, defense parameters, model parameters, and training data. The adversary has to just query the robust model and train one or more substitute models. We also evaluated two other black-box attack against the aforementioned defenses, but they performed poorly in comparison to our presented attack. We suggest that other researchers use our approach for benchmarking models in a black-box setting in cases where a defense prevents the attacker from calculating useful gradients from the target model.

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 7: The model accuracy and average l2 norm of required perturbation across different threshold

Figure 9: Adversarial examples crafted with different λ for MNIST

Figure 10: Adversarial examples crafted with different λ for CIFAR-10

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] 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

[4] 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.

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

[6] 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.

[7] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Chaowei Rahmati, Amir 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.

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

[9] 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.

[10] 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.

[11] Konstantinos Kamnitsas, Enzo Ferrante1, Sarah Parisot1, Christian Ledig, Aditya Nori, Daniel Criminisi, Antonio Rueckert, and Ben Glocke. 2016. DeepMedic for Brain Tumor Segmentation. In *Proceedings MICCAI-BRATS Workshop*. 18–22.

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

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

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

[15] 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*.

[16] 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)*.

[17] 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*.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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

[23] 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.

[24] 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

[25] 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).

[26] 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. Human-level control through deep reinforcement learning. *Nature* 529 (2016), 484–489.

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

[28] 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).

[29] Tuan A. Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. 2016. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 258–263.

[30] 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*.

[31] 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.

[32] 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.

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

APPENDIX

Some of 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 8: The model architecture and hyper parameters used for training model for RFN evaluation

Index of data	A	B	C	D
[0-10000)	$[-\frac{2}{255}, \frac{2}{255}]$	$[-\frac{3}{255}, \frac{3}{255}]$	$[-\frac{4}{255}, \frac{4}{255}]$	$[-\frac{5}{255}, \frac{5}{255}]$
[10000-20000)	$[-\frac{4}{255}, \frac{4}{255}]$	$[-\frac{6}{255}, \frac{6}{255}]$	$[-\frac{8}{255}, \frac{8}{255}]$	$[-\frac{10}{255}, \frac{10}{255}]$
[20000-30000)	$[-\frac{6}{255}, \frac{6}{255}]$	$[-\frac{9}{255}, \frac{9}{255}]$	$[-\frac{12}{255}, \frac{12}{255}]$	$[-\frac{15}{255}, \frac{15}{255}]$
[30000-40000)	$[-\frac{8}{255}, \frac{8}{255}]$	$[-\frac{12}{255}, \frac{12}{255}]$	$[-\frac{16}{255}, \frac{16}{255}]$	$[-\frac{20}{255}, \frac{20}{255}]$
[40000-50000)	$[-\frac{10}{255}, \frac{10}{255}]$	$[-\frac{15}{255}, \frac{15}{255}]$	$[-\frac{20}{255}, \frac{20}{255}]$	$[-\frac{25}{255}, \frac{25}{255}]$
[50000-60000)	$[-\frac{12}{255}, \frac{12}{255}]$	$[-\frac{18}{255}, \frac{18}{255}]$	$[-\frac{24}{255}, \frac{24}{255}]$	$[-\frac{30}{255}, \frac{30}{255}]$
[60000-70000)	$[-\frac{14}{255}, \frac{14}{255}]$	$[-\frac{21}{255}, \frac{21}{255}]$	$[-\frac{28}{255}, \frac{28}{255}]$	$[-\frac{35}{255}, \frac{35}{255}]$
[70000-80000)	$[-\frac{16}{255}, \frac{16}{255}]$	$[-\frac{24}{255}, \frac{24}{255}]$	$[-\frac{32}{255}, \frac{32}{255}]$	$[-\frac{40}{255}, \frac{40}{255}]$

Table 9: The range of random noise added to different part of dataset and for different model 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 10: The model architecture used for evaluating 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 11: 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 12: 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 13: 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 14: The model 3 architecture

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

Table 15: The model 4 architecture