UNIVERSITY OF CALIFORNIA,
IRVINE

ADMIT: An Adversarial Defense Methodology for Neural Networks based on Randomization and Reconstruction

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Engineering

by

Marzieh Ashrafiamiri

Dissertation Committee:
Professor Fadi Kurdahi, Chair
Professor Nader Bagherzadeh
Professor Nikil Dutt

2021

DEDICATION

I am dedicating this thesis to four beloved people who have meant and continue to mean so much to me. A special feeling of gratitude to my loving mom, Vajiheh, whose words of encouragement and push for tenacity ring in my ears. Also, my loving dad, Mohammadreza, though his life was short, I will make sure his memory lives on as long as I shall live.

I also dedicate this dissertation to my brother, Mohsen who has always loved me unconditionally and is very special to me.

This thesis is specially dedicated to my husband, Amir Hosein, who has never left my side and has been a constant source of support and encouragement during the challenges of graduate school and life.

I am genuinely thankful for having you all in my life.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor, Professor Fadi Kurdahi, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would also like to thank my committee members, Professor Nader Bagherzadeh and Professor Nikil Dutt, for assigning their valuable time to review my work.

I would like to offer my special thanks to my tutors, Dr. Sai Manoj Pudukotai Dinakarrao, Minjun Seo, and Mohammed Fouda, for their valuable guidance throughout my studies. You provided me with the tools that I needed to choose the right direction and successfully complete my dissertation.

I thank ACM/IEEE for allowing me to use the content of our original paper [1] published in Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD). Some text of this thesis is a reprint of the material as it appears on MLCAD'20.

Additionally, I want to thank UCI's Department of Electrical Engineering and Computer Science for giving me the initial fellowship funding, which enabled me to prosper on this glorious path.

Finally, I would like to thank the National Science Foundation (NSF) for partially funding my work under award number CCF-1704859. Any opinions, findings, conclusions, or recommendations expressed in this thesis are those of the author and do not necessarily reflect the views of the funding agency.

VITA

Marzieh Ashrafiamiri

EDUCATION

Master of Science in Computer Engineering **2018-2021**
University of California, Irvine *Irvine*

Bachelor of Science in Computer Engineering **2013-2018**
Sharif University of Technology *Tehran*

RESEARCH EXPERIENCE

Graduate Research Assistant **2018–2021**
University of California, Irvine *Irvine*

Research Assistant and Data Analyst **2017–2018**
Sharif University of Technology *Tehran*

TEACHING EXPERIENCE

Teaching Assistant **2018–2021**
University of California, Irvine *Irvine*

Teaching Assistant **2015–2018**
Sharif University of Technology *Tehran*

REFEREED JOURNAL PUBLICATIONS

NEWERTRACK: ML-Based Accurate Tracking of In-Mouth Nutrient Sensors Position Using Spectrum-Wide Information **2020**
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

REFEREED CONFERENCE PUBLICATIONS

R^2AD: Randomization and Reconstructor-based Adversarial Defense for Deep Neural Networks **2020**
ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)

ABSTRACT OF THE DISSERTATION

ADMIT: An Adversarial Defense Methodology for Neural Networks based on Randomization and Reconstruction

By

Marzieh Ashrafiamiri

Master of Science in Computer Engineering

University of California, Irvine, 2021

Professor Fadi Kurdahi, Chair

From simple time series forecasting to computer security and autonomous systems, machine learning (ML) is employed in a wide range of applications. Despite the fact that machine learning algorithms are resistant to random noise, it has been shown that intentionally targeted perturbations to the input data, known as *adversarial samples*, can lead to a significant degradation in the ML performance. Existing countermeasures to mitigate or minimize the impact of adversarial samples, including adversarial training or randomization, are limited to specific categories of adversaries, are computationally costly, and/or result in lower performance even when no adversaries are present. To address the inadequacies of the existing works on adversarial defense, we propose a two-stage adversarial defense technique (*ADMIT*). To thwart the exploitation of the deep neural network by the attacker, we first include a random nullification (RNF) layer. The RNF nullifies/removes some of the features from the input randomly to lessen the influence of adversarial noise and minimize the attacker's capability of extracting the model parameters. The elimination of input features using RNF, on the other hand, reduces the performance of the ML. We outfit the network with a Reconstructor as an antidote. The Reconstructor primarily contributes to reconstructing the input data by utilizing an autoencoder network, but based on the distribution of the normal samples, thereby improving the performance, and also being robust to the

adversarial noise. We evaluated the performance of proposed multi-stage *ADMIT*on the MNIST digits and Fashion-MNIST datasets against variety of adversarial techniques including FGSM, JSMA, BIM, Deepfool, and CW attacks. Our findings report improvements as high as 80% in the performance when compared to the existing defenses such as adversarial training and randomization-based defense.

Chapter 1

Introduction

Application of machine learning (ML), especially deep learning, has been widely adopted in a plethora of applications such as image processing [2], hardware security [3, 4, 5, 6], health monitoring [7, 8, 9, 10], natural language processing [11] and autonomous systems [12]. Deep learning has showcased an appreciable performance in these applications and has also shown resilience against random noise in the input. Despite robustness against random noise, the recent works have shown that the machine learning techniques are vulnerable to specially crafted perturbations, adversarial samples [13, 14, 15, 16, 17]. The perturbed samples include a minimal amount of noise that can not even be seen by human's naked eyes nor affect the recognition capability by naked human eyes, but can be misclassified by the ML classifiers.

Multiple countermeasures have been proposed in the literature to alleviate the misclassification induced by adversarial perturbations. The following are some of the most common defenses (the more detailed explanation of these techniques can be found in Section 2.2):

- adversarial training [18] - train the classifier with adversarial examples
- adversarial sample detection [19, 20] - detect the normal and adversarial samples

- defensive distillation [21] - blocks the attacker from obtaining the loss gradient, thus, decreasing the feasibility to attack
- random nullification features [22] - utilize random nullification layer as a defense against adversaries
- digitization [23] - techniques such as thermometer encoding to minimize the sensitivity exploitation.

Despite the elevated robustness against the adversarial samples, these techniques suffer from setbacks. Adversarial training, for example, is limited to a single attack; techniques like random nullification features and digitisation, while successful against adversaries, perform poorly on normal (benign) samples. We perform a case study to explain how random nullification affects various adversarial attacks. Section 4.1 discusses the experimental details and network parameters. However, in this experiment, we use randomized inputs for both the training and testing phases (for further information, see Section 3.1).

Table 1.1: Comparison of neural network performance with and without randomization

	MNIST digits dataset	
	clean dataset	FGSM (ϵ=0.3)
Accuracy of Network	99.21%	27.77%
Accuracy of network with traditional random nullification	98.87%	66.36%

As a case study, we have implemented the random nullification layer as the pre-processing of the input to alleviate the impact of adversaries. We have implemented fast sign gradient method (FGSM) attack to craft the adversaries. The hyper-parameter required for FGSM is set to 0.3 for this case study. Despite the fact that the randomization strategy helps in eluding the adversarial attacks, performance degradation is found in the case of normal samples, as shown in Table 1.1. As a result, it is of a dire need to devise a technique that is effective against adversarial attacks while also maintaining performance when standard samples

are used. To address the inadequacies of existing works and address the challenges as outlined above, this work introduces *ADMIT* - randomization+reconstructor-based adversarial defense.

The proposed *ADMIT* is a multi-stage approach, where the random nullification layer (RNF) is initially designed to randomly remove the pixels from the input samples (normal or adversarial), hence reducing the impact of adversarial noise. However, such a random removal of pixels (input features) might damage normal/benign pixels, resulting in performance degradation. To mitigate such degradation, we introduce an autoencoder-based reconstructor network, which aids in reconstructing the input (in this case, an image) with the same distribution as normal samples. Using autoencoders to reconstruct samples and/or feature selection has been a commonly popular technique in various applications of machine learning such as motion tracking [24], hardware security [25], and health monitoring [26, 27, 28]. It's worth mentioning that the reconstructor network is trained with benign data, thus reconstructs the input whose data distribution is close to the original/benign samples.

We evaluated the proposed *ADMIT* on MNIST digits [29] and Fashion-MNIST [30] datasets with various attacks ranging from FGSM [14] to recently proposed Carlini & Wagner (C&W) [31] attacks. The rationale to showcase for this dataset is that this is one of the simplest datasets that can be adversarially perturbed even with minimal noise when compared to other data.

The contributions of this work can be outlined in a three-fold manner as follows:

- We introduce a random nullification layer as a part of data pre-processing to decrease the impact of adversarial perturbations and reduce the likelihood of attackers obtaining the classifier model.
- An autoencoder-based reconstructor is introduced during inference for reconstructing input and eventually resulting in an improved performance against both normal and

adversarial samples.

- Accuracy improvement for both adversarial and normal samples along with minimized adversarial impact through the proposed *ADMIT*.

The rest of this work is organized as follows: Section 2 introduces the relevant background regarding the existing adversarial attacks and defenses against these attacks. Section 3 describes the proposed *ADMIT* with more information on the design of the RNF layer and the reconstructor network. The evaluation of proposed *ADMIT* is presented in Section 4 with conclusions drawn in Section 5.

It is important to note that this thesis is based on our original paper [1] published in Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD). Thus, it contains information of the our original paper.

Chapter 2

Background and Related Work

Machine learning and Deep learning algorithms have revolutionized the Artificial Intelligence domain having demonstrated impressive performance on applications ranging from computer vision and hardware security to health monitoring. However, these state of the art, pattern recognition classifiers are easily fooled by adversarial samples.

Adversarial samples are often generated by adding carefully crafted noise perturbations to the normal input samples. This introduced noise leads to misclassification by the underlying ML classifier or predictor. It needs to be noted that care is often taken to optimize the difference between normal and adversarial samples.

In this section, we present different techniques widely used for generating the adversarial samples, and review some of the popular defense techniques deployed.

2.1 Existing Adversarial Attacks

Adversarial samples are the samples that are generated by introducing crafted perturbations into the normal input data generated by introducing optimum yet worst-case perturbations in order to make the adversarial data look similar to the normal input data, but still the ML model mispredicts the class with a high probability. These adversarial samples can be considered as an optical illusion for the ML classifiers.

Here, we present fast gradient sign method (FGSM) technique. Other techniques such as Basic Iterative Method (BIM), Jacobian Saliency Map Attack (JSMA) and Carlini & Wagner (C&W) can be seen as advanced and sophisticated versions of FGSM. Moreover, information regarding the Deepfool attack is also presented.

2.1.1 Fast Gradient Sign Method (FGSM)

Fast Gradient Sign Method (FGSM) is one of the first and most basic approaches for generating the adversary [14]. The main idea of FGSM is to perturb the input (image) with the gradient of the loss w.r.t. the input (image) data. Further, increasing the magnitude of the added noise or perturbation until the input (image) is misclassified by the underlying ML model. The complexity of generating FGSM attack is lower compared to other adversarial attacks. Some of the advantages of this technique are its low complexity and fast implementation.

Consider an ML classifier model with θ as the hyper-parameter with x and y being the input and corresponding output, respectively. The cost function is represented by $L(\theta, x, y)$. Then the perturbation with FGSM is computed as the sign of the model's cost function gradient.

The adversarial perturbation generated with FGSM [14] is mathematically given as

$$x^{adv} = x + \epsilon sign(\nabla_x L(\theta, x, y)) \tag{2.1}$$

where ϵ is a scaling constant between 0.0 to 1.0 is set to be very small such that the variation in x (δx) is undetectable. One can observe that in FGSM the input x is perturbed along each dimension in the direction of the gradient by a perturbation magnitude of ϵ. Considering a small ϵ leads to well-disguised adversarial samples. The difference between the original input and the adversarial sample is generally undetectable to the human eye. Also, a large ϵ, is likely to introduce large perturbations.

2.1.2 Basic Iterative Method (BIM)

As one can observe from (2.1) that the noise is added to every feature of the input in the case of FGSM and no optimization is performed. To address this, Kurakin proposed an iterative version of FGSM, termed as Basic iterative method (BIM) in [32]. In BIM, instead of applying the adversarial perturbation once with ϵ, the perturbation is iteratively applied by incrementing the perturbation i.e., ϵ. This leads to gradual increment in the added perturbation (noise) to the input and terminates as soon as the misclassification requirements are met.

The adversarial perturbation generated with BIM can be mathematically defined as:

$$\begin{aligned} x_0^{adv} &= x \\ x_{N+1}^{adv} &= Clip_{x,\epsilon}(x_N^{adv} + \epsilon sign(\nabla_x L(\theta, x_N^{adv}, y)) \end{aligned} \tag{2.2}$$

Here, $Clip_{x,\epsilon}$ represents the clipping of the adversarial input magnitudes such that the adersarial samples are close (similar) to the original samples x. Thus, BIM allows higher flexibility to input the optimal perturbations compared to the FSGM, as the perturbations that are added can be controlled and the distance of the adversarial sample from the classification boundary (manifold) can be carefully fine-tuned.

The work in [32] shows that the BIM can lead to cause higher misclassifications compared to the FGSM on the Imagenet samples.

2.1.3 Jacobian-based Saliency Map Attack (JSMA)

In contrast to applying noise to every single feature of the input data, [21] proposes an iterative technique to add the perturbation, where the forward derivative of DNN is exploited for adding the perturbations.

Consider a neural network F with input x. If the corresponding output is class j, we represent the model as $F_j(x)$. The main principle of this work is: to provide target t as the output, the probability for $F_t(X)$ must be increased, and simultaneously, the probabilities of $F_j(X)$ for all the other classes i.e., $j \neq t$ have to be decreased, until $t = arg\ max_j F_j(X)$ is achieved.

This is accomplished by exploiting the saliency map, as defined below

$$S(X,t)[i] = \begin{cases} 0, \text{if} \frac{\partial F_t(X)}{\partial X_i} < 0 \text{ or } \sum_{j \neq t} \frac{\partial F_j(X)}{\partial X_i} > 0 \\ (\frac{\partial F_t(X)}{\partial X_i}) | \sum_{j \neq t} \frac{\partial F_j(X)}{\partial X_i} |, \text{otherwise} \end{cases} \tag{2.3}$$

For an input feature i starting with the normal input x, we determine the pair of features $\{i, j\}$ that maximizes $S(X,t)[i] + S(X,t)[j]$ and perturb each of the features by a constant offset ϵ. This process is repeated iteratively until the target misclassification is achieved. It

is worth-mentioning that JSMA is a targeted attack, however it can be used as an untargeted attack as well.

2.1.4 Carlini and Wagner Attack (CW)

One of the most recent adversarial attacks is introduced by Carlini and Wagner in [31], popularly called as Carlini and Wagner (CW) attack. The CW attack is shown to outperform adversarial defense techniques such as defensive distillation. It is an iterative attack that finds adversarial samples against multiple defenses as compared to other attacks. At a high level, this attack is iterative using Adam optimizer and a specially chosen loss function to find adversarial examples with lower distortions than the other attack, which imposes the cost of complexity and causes the attack to be much slower than the other attacks.

It encompasses a range of attacks based on the norms, all cast through the same optimization framework, thus resulting in 3 powerful attacks, that are designed employing L_0, L_2, and L_∞ norms.

For the L_2 attack, which is considered in this work, the perturbation in the input i.e., δ is defined in terms of an auxiliary variable ω. The objective of the CW attack with L_2 norm can be mathematically defined as

$$\delta_i^* = \frac{1}{2}(\tanh(\omega_i + 1)) - x_i \tag{2.4}$$

Then, the δ_i^* which is an unrestricted perturbation is optimized over ω as follows:

$$\min_{\omega} ||\frac{1}{2}(\tanh(\omega)+1) - x||_2^2 + cf(\frac{1}{2}\tanh(\omega)+1) \tag{2.5}$$

Similarly, if the L_2 is considered, the optimization becomes

$$\min_{\delta} ||\delta||_2 + c \cdot f(x+\delta) \tag{2.6}$$

$$S.T. x + \delta \in [0,1]^n \tag{2.7}$$

where f (objective function) is defined as

$$f(x') = \max(\max\{Z(x') : i \neq t\} - Z(x') - k) \tag{2.8}$$

Here, $Z(x')$ is the pre-softmax output for class i, t is the target class, and k is the parameter that controls the confidence with which the misclassification occurs. The parameter k encourages the solver to find an adversarial instance x' that will be classified as class t with high confidence.

The three variants of this attack were shown to be quite effective in generating adversaries compared to other attacks discussed previously.

2.1.5 DeepFool

DeepFool (DF) is an untargeted adversarial attack optimized for L_2 norm, introduced in [33]. DF is an efficient adversarial attack that is capable of producing adversarial samples that highly resemble the original inputs as compared to the aforementioned adversarial samples, especially FGSM and BIM attacks discussed earlier. The principle of the Deepfool attack is to assume neural networks as completely linear with a hyperplane separating each class from another.

Based on this assumption, if you consider a linear binary classifier, that the robustness of the model (f) for an input x_0 is equal to the distance of x_0 to the hyperparameter plane (which seperates the 2 classes). Minimal perturbation to change the classifier's decision corresponds to the orthogonal projection of x_0 onto the hyperparameter plane. The perturbed sample is given by:

$$x_{i+1} = x_i + (-\frac{f(x_i)}{||\nabla f(x_i)||_2^2} * \nabla f(x_i)) \tag{2.9}$$

This process is repeated multiple times for creating the adversaries. This process is terminated when an adversarial sample is found, i.e., misclassification happens.

With multiclass classifiers, let's say the input is x and for each class there is a hyperplane (straight plane that divides one class from the others) and based on the place in the space where x lies it is classified into a class. All this algorithm does is first, it finds the closest hyperplane, and then projects x onto that hyperplane and pushes it a bit beyond with adding the minimal perturbation possible. The loop will continue until minimal perturbation is found to cause misclassification.

2.2 Existing Adversarial Defense

Till now, different adversarial attack techniques are discussed. Here, we discuss some of the prominent existing defenses against different adversarial attacks.

2.2.1 Adversarial Training

Adversarial training is one of the preliminary solutions for making the ML classifiers robust against the adversarial examples, proposed in [18]. The preliminary idea is to train the ML classifier with the adversarial examples so that the ML classifier can have adversarial information [13, 14, 33] and adapt its model based on the learned adversarial data. One of the major drawbacks of this technique is to determine what kind of attack is going to happen and train the classifier based on those attacks and determining the criticality of the adversarial component.

2.2.2 Defensive Distillation

Defensive distillation is another defense technique proposed in [21], that trains the classifier using the distillation training techniques and hides the gradient between the softmax layer and the pre-softmax layer. This makes it complex to generate adversarial examples directly on the network [34], as the knowledge is imparted from a bigger network during the training process. However, [31] shows that such a defense can be bypassed with one of the following three strategies: (1) choosing a more proper loss function; (2) calculation of gradients from pre-softmax layer rather than softmax layer; or (3) attack an easy-to-attack dummy network first and then transfer to the distilled network, similar to the distillation defense. The generation of adversaries can be more straightforward if the attacker knows the defense network's parameters and architecture, i.e., white-box attack.

2.2.3 MagNet

MagNet is proposed in [35], where a two-level strategy with detector and reformer is proposed. In the detector phase(s), the system learns to differentiate between normal and adversarial examples by approximating the manifold of the normal examples. This is performed with the aid of autoencoders. Further, in the reformer, the adversarial samples are moved close to the manifold of normal samples with small perturbations. Further, using the diversity metric, the MagNet can differentiate the normal and adversarial samples. The MagNet is evaluated against different adversarial attacks presented previously and have shown to be robust in [35].

2.2.4 Randomization

Randomization based defenses can be seen effective against the evasive adversarial attacks, as they try to hide the model or lead to minimal data retrieval for crafting an adversary. For instance, the work in [22] introduces a randomization feature nullification at both training and testing phases to make the DNN models robust against the adversarial samples. This randomization can be seen as a special case of drop-out techniques. Even if the adversarial attackers manage to obtain the model, the use of randomly nullifying the features try to reduce the adversarial impact on the network by removing the features. Despite, randomization aids in combating adversaries, a performance loss is seen for normal samples, i.e., samples without adversaries, as shown in Table 1.1.

2.2.5 Classifier Robustifying

In [36] using Bayesian classifiers as a way of designing a robust architecture of DNNs is explained. Gaussian precessors are used to reveal the secret variables as parameters of the

gaussian distribution. These are then encoded with the help of RBF Kernels. They proposed the GPs combined with RBF kernels called GP hybrid deep neural networks (GPDNNs). They have mentioned that GPDNNs are more robust against adversarial examples.

2.2.6 Network verification

Understanding whether an input satisfies or violates properties of a neural network is one way of defending against adversarial attacks. Network verification checks the properties to detect the new attacks. One way of neural network verification is Reluplex, which uses Relu function [37]. There was shown in the paper that adversarial data with a little percentage of perturbation would not cause any misclassification for the network. There exists a problem with Reluplex method. Due to its massive computation, it is very slow and just works fine for simple networks with just a few hundred nodes.

2.2.7 Detecting Adversaries

Another idea of defense proposed in the existing works is to detect adversarial examples with the aid of statistical features [20] or separate classification networks [19]. In [19], for each adversarial technique, a DNN classifier is built to classify whether the input is a normal sample or an adversary. The detector was directly trained on both normal and adversarial examples. The detector showed good performance when the training and testing attack examples were generated from the same process, and the perturbation is large enough. However, it does not generalize well across different attack parameters and attack generation processes.

In contrast to the existing works, the proposed *ADMIT* utilizes the concepts of randomization to prevent the attacker from obtaining the model and reducing the impact of adversaries

through random nullification features. To address the prevailing challenge of performance degradation for normal samples after random nullification, the *ADMIT* utilizes a reconstructor network, i.e., autoencoder that reconstructs the input after randomization, thereby retaining the performance. It needs to be noted that the reconstructor rebuilds the input after randomization. However, special care is taken to ensure that adversaries are not reconstructed by training the reconstructor with the data distribution of original (not adversarial) data. Thus, the proposed *ADMIT* is capable of minimizing the impact of adversaries through random nullification technique and can retain the performance through autoencoder-based input reconstruction.

Chapter 3

Design *ADMIT*

In this study, we propose a network that consists of two types of layers: a random nullification layer termed as RNF and an autoencoder-based reconstructor, as shown in Figure 3.1. RNF is a layer that simply omits some features of our input data. The reconstructor layer assists the network in classifying images. In the following sections, we will describe these two layers. It needs to be noted that the reconstructor layer has the duty of denoising and rebuilding images; it just helps us to show the strength of our proposed network.

Based on the Figure 3.1, the defense proposed as *ADMIT* includes two phases, training phase and testing phase. During the training phase, we utilize the Random Nullification layer to train a neural network. Moreover, an auto-encoder based reconstructor is being trained using clean and manually-added-noise data. In the testing phase, first, the reconstructor network is used to create reconstructed samples of test data. For the next step, by using the calculated reconstruction error, *ADMIT* will decide if the data point is unreconstructable or not. Finally, each data point that is considered reconstructable will be classified by the RNF-trained network.

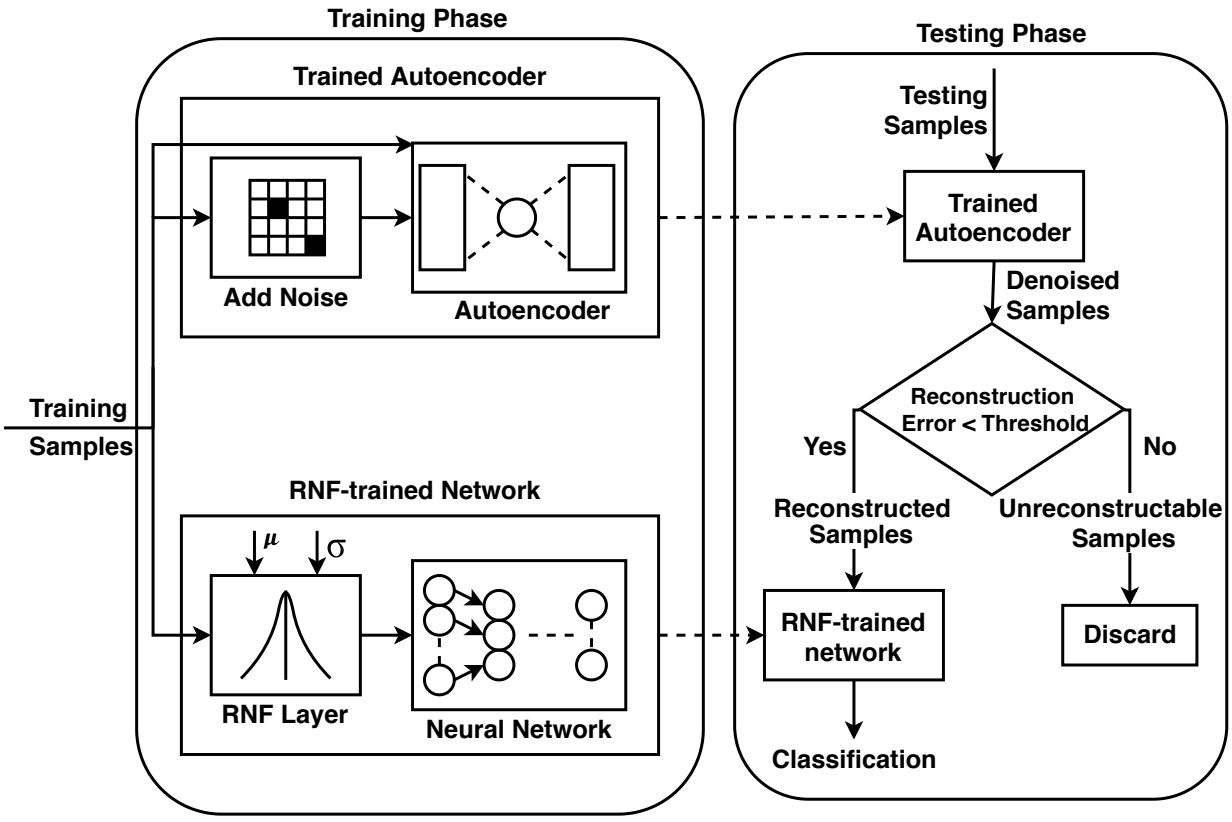


Figure 3.1: Proposed Defense Network

3.1 Random Nullification Layer (RNF)

One of the efficient ways to defend against adversarial attacks is having random nullification layer, as described in [22]. In simple terms, this defensive strategy involves adding an extra layer, called RNF, to the neural network. The RNF layer, with the responsibility of nullifying some features of input data, is located between the inputs and the first hidden layer in the training phase (this location is selected among all possibilities). Suppose we have some images as training inputs for our method. The RNF layer will randomly omit some of the pixels of the data points, meaning that it will set them to zero and lead them to the trained network. The number of features that are nullified in each input data is specific and follows a Gaussian distribution.

Hence, the first step is to choose the parameters of Gaussian distribution and calculate the number of features in each input which are going to be nullified. Based on the number calculated beforehand, some features are selected from a uniform distribution and are being nullified. The worth mentioning aspect of this method is that the probability of each feature chosen to be nullified is the same. So, for each input sample x, the I^x should be made which is the same size of input and have random zeros and ones in it. Then, the x^* is product of these arrays, as shown below.

$$x^* = x \bigodot I^x \tag{3.1}$$

Randomly removing pixels helps to reduce the impact of adversaries, thus added noise's impact is reduced in the existing attacks as shown in Figure 3.2. However, a challenge with having RNF is performance degradation, even on normal samples, due to the randomness of nullifying features which may delete some required features. We will practically demonstrate that gaining accuracy in deleting noise of adversarial example is much notable than losing

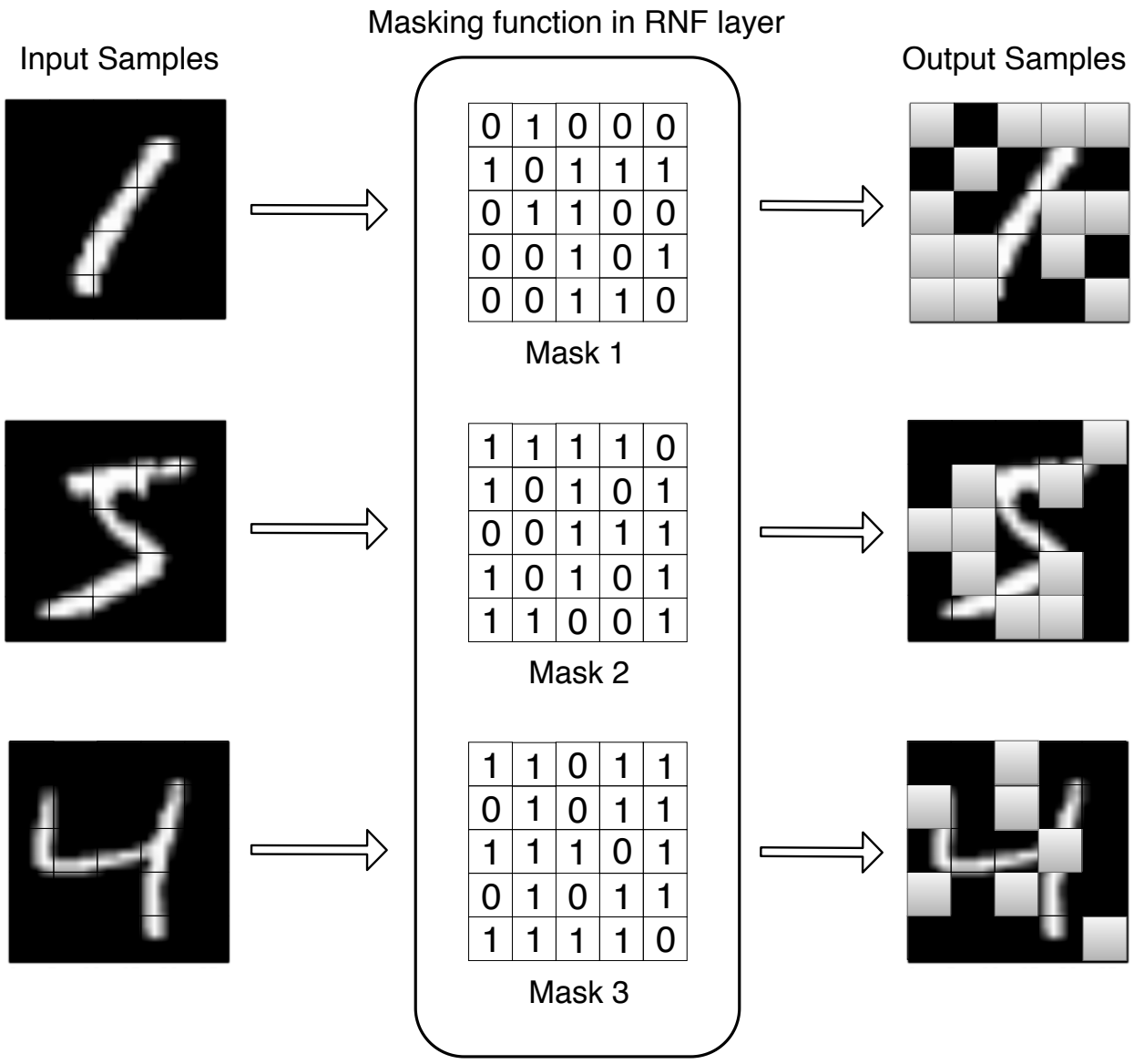


Figure 3.2: Process of nullification i.e., impact of three different RNF masks.

it on normal samples.

There exist different feasible choices regarding when the RNF layer can be deployed: (1) only in training phase, (2) only in testing phase, and (3) in both phases. We analyze the impact of utilizing the RNF on these different choices. Table 3.1 demonstrates the results based on the location when the FGSM attack is implemented on MNIST digits data. As seen, using the RNF layer in both training and testing phases leads to the best performance compared to other scenarios. In this scenario, the model is adapted to feature nullification and when deploying the removing pixels to the testing phase, the model will easily obtain better performance.

Table 3.1: Results for different locations of RNF

Utilization of RNF	Accuracy for FGSM ϵ=0.3
Only training	10.02%
Only testing	29.69%
Training and Testing	66.36%

From all the possible choices to utilize the RNF layer, we propose the one that is used for training the network. However, it has lower performance than other scenarios. The power of RNF just in training will be flourished by using the reconstructor.

In the chosen scenario, first we will use RNF layer on our training set, means clean MNIST (both MNIST digits and Fashion-MNIST in different experiments). Then, we will train the network. Due to the training phase, our model is now adapted to considering some noise/nullification. Since dealing with adversarial data is the purpose of the network in the testing phase, the model trained with RNF shows better performance when reconstructor is added. In the Section 4, the Table 4.2 witness the statement. As aforementioned, while passing through the RNF layer, the pixels of the input data will be randomly nullified/omitted. These pixels chosen to be omitted are specific to one input data, and their coordinations are different among different data points. When all the data points pass through the RNF layer,

the nullified data points are being used for the training phase. Hence, in all of the epochs, the nullified pixel indices are similar.

3.2 Reconstructor

One way of defending against attacks is determining the adversarial input and trying to denoise it. The noticeable advantage of the proposed function is attack-independent, unlike the previous methods like adversarial training. In other words, having just normal data is sufficient for making a reconstructor.

The reconstructor first calculates the distance between the input sample (adversarial or normal) and the denoised sample, as presented in Equation (3.2). Further, depending on the specified threshold, it determines whether the given sample is normal or adversarial. The threshold should neither be very large to misunderstand adversarial input to normal data nor be very small to increase the probability of deciding the normal sample as untrusted data.

For the implementation of the reconstructor layer, autoencoders are used, which consists of an encoder and a decoder as shown in Figure 3.3. For training the autoencoder, some random noise should be added to a clean dataset (these are not adversarial data). The noise factor we choose is equal to 0.1.

An ideal autoencoder should not change the classification result of normal data. Encoders are used in learning a latent representation of data, and decoders use these features for the reconstruction of data. Although in the case of normal inputs, the reconstruction error will be low, for adversarial inputs it leads to a high amount of error. There exist many ways to measure the reconstruction error between data and normal samples which L^1 and L^2 norms are chosen practically. For these p-norms errors, increasing p will increase the importance

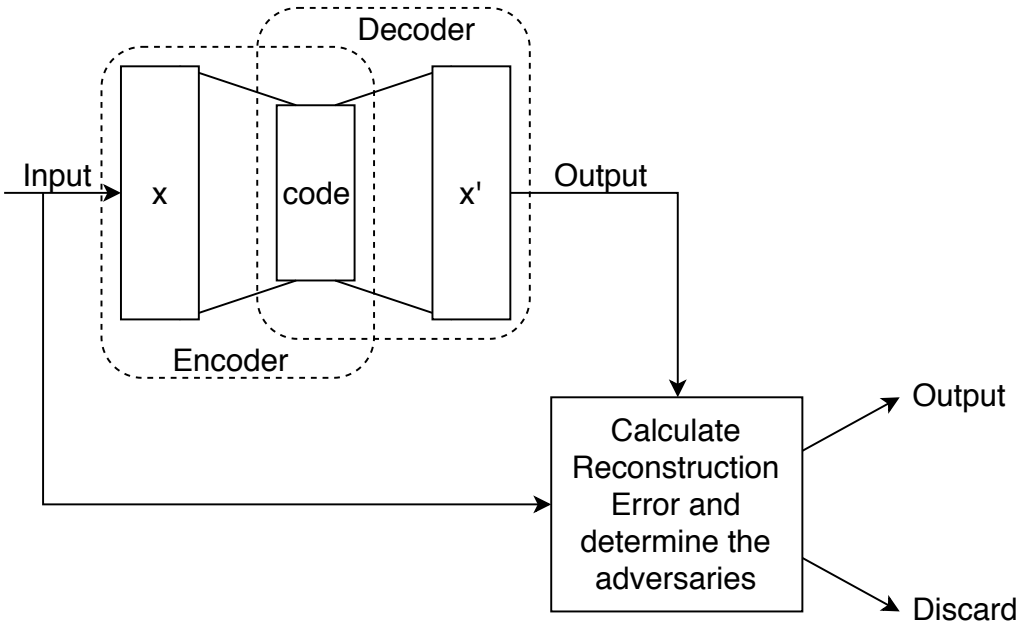


Figure 3.3: Design of reconstructor

of maximum distance between all pixels, but decreasing p will lead to more attention on each pixel itself. If we consider $ae(x)$ be the output of autoencoder for the input x, the reconstruction error is then calculated by (3.2).

$$E(x) = ||x - ae(x)||_p \tag{3.2}$$

Based on the threshold obtained from the clear and normal input sample data, the reconstructor will decide if a sample is labeled as reconstructable or unreconstructable data. The unreconstructable sample shows so much noise that the actual data is unrecognizable. It is one of our goals to intercept such samples and prevent them from moving forward, due to the potential of downgrading the performance. All less noisy data (normal or adversarial) will then go through the trained network to be classified.

Among the low noisy forwarded inputs, some are normal, and some are adversarial. Normal data as inputs should remain unmodified after going through the autoencoder. On the other hand, the autoencoder will attempt to denoise the adversarial inputs so that they

resemble normal data, and the adversarial noise is filtered out. Using this technique have two important advantages. First, because the normal data remains the same, the method will not change the classification accuracy. Also, the classification accuracy of adversarial samples will increase because of denoising.

Chapter 4

Simulation Analysis

4.1 Experimental Setup

The performance of the proposed *ADMIT*is evaluated against multiple adversarial attacks (FGSM [14], BIM [32], JSMA [21], DeepFool [33] and Carlini and Wagner (CW) [31]). The experiments are executed on the Intel Xeon E3-1225 V5 processor with 16 GB RAM and also NVIDIA GeForce GTX 1660. We evaluated the MNIST digits dataset with 50000 images utilized for training the neural network architecture and 10000 utilized for testing, i.e. 80-20% data split for training and testing. We utilized Tensorflow library [38] to implement and evaluate our proposed *ADMIT*. Also, we have employed Cleverhans library [39] to craft the adversaries and evaluate them. For evaluation, we considered whitebox and blackbox scenarios, i.e. attack generation and testing on the same network termed as whitebox attack scenario and different networks for generation and testing for blackbox scenario. The architectural details of the neural network used for *ADMIT* evaluation of MNIST digits are presented in Table 4.1. The accuracy of classifying the MNIST digits dataset using the network presented in Table 4.1 is 99.21%.

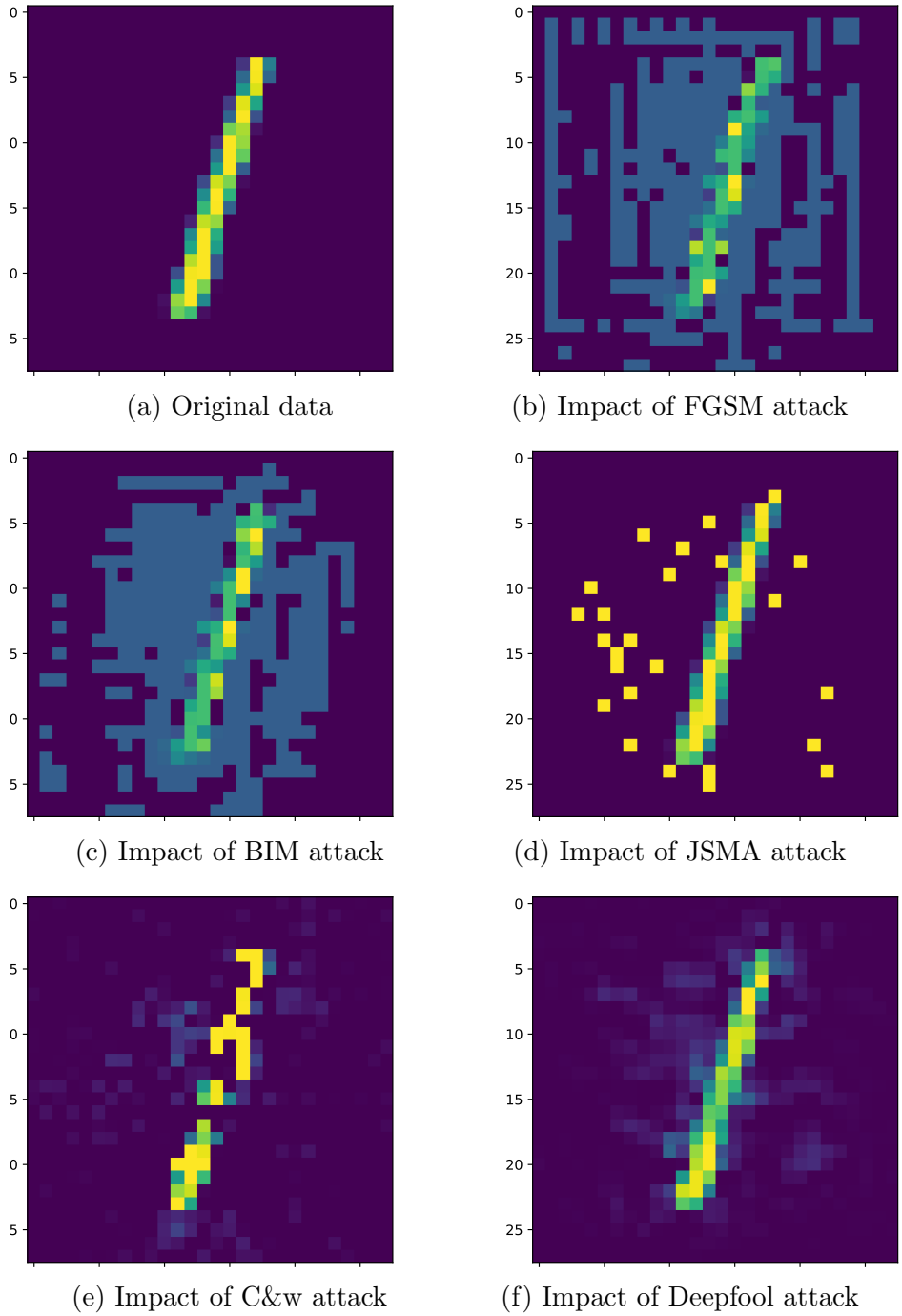
Table 4.1: Architecture of network and reconstructor for MNIST digits dataset

classifier		reconstructor		
Conv+Relu	32×3×3	encoder	Conv+Sigmoid	1×3×3
Conv+Relu	64×3×3		Average Pooling	2×2
Max Pooling	2×2		Conv+Sigmoid	3×3×3
Dense+Relu	128	decoder	Conv+Sigmoid	3×3×3
Dense+softmax	10		Up Sampling	2×2
			Conv+Sigmoid	3×3×3
			Conv+Sigmoid	1×3×3

As explained in Section 3, the reconstructor is made of an autoencoder. The details of autoencoder's architecture, which is used in the experiment of the MNIST digits dataset, can be found in Table 4.1. After denoising each sample, the reconstructor will calculate the reconstruction error. Samples with a lower error than the threshold will then go through the RNF trained network. The RNF layer, which inherits features of a normal distribution, needs some parameters to be determined before. The two most important parameters are Mean and standard deviation, which are 3 and 4 for the experiments.

4.2 Performance Against Different Attacks

As mentioned earlier, we evaluate the impact of adversarial attacks against the proposed *ADMIT*. First, let us consider the case of the blackbox scenario, where the attacker has no information regarding the network for which attack is crafted, i.e. the attacker relies on the transferable properties of the attack. Similarly, we have considered the case of the whitebox scenario, i.e. the attacker has the information regarding the classification network and can generate the attack based on that information.



(a) Original data

(b) Impact of FGSM attack

(c) Impact of BIM attack

(d) Impact of JSMA attack

(e) Impact of C&w attack

(f) Impact of Deepfool attack

Figure 4.1: The comparison between original MNIST image and generated adversarial data

As the first step towards experiments, by using Cleverhans library, we generated required adversarial data of different attack. Table 4.1 shows impact of different adversarial attacks on MNIST image. The parameters for generating each of the illustrated images are equal to the parameters in Table 4.2.

4.2.1 Evaluation with MNIST Digits Dataset

Table 4.2 presents the performance of the different attacks of MNIST digits data on the network defined in Table 4.1. The attack parameter like $\epsilon \in [0, 1]$ for FGSM indicates the amount of perturbation added to the input. For the scenario of blackbox attack: as one can observe that without any defense, the performance of classification reduces to 27.7%. Similarly, when only utilizing Random Nullification Layer (RNF) as defense [22], the performance reduces to 10.02%, where the loss is due to the loss of information. Similarly, we replicated the work presented in 3 where the reconstructor is used. As seen, the utilizing of the RNF also improved the performance against all attacks, when using with the reconstructor. The rationale for this is that the FGSM is similar to a brute force kind of approach in terms of adding the noise and can be easily circumvented compared to the rest of the adversarial attacks.

In the case of BIM attack, as it can be seen clearly from Table 4.2, the attack samples are transferable - as the accuracy drops to 23.13% on an average when no defense is deployed. Similar behavior is observed in this case, which is an iterative version of FGSM. Compared to the simple usage of the RNF layer [22] or utilizing reconstructor [35], the proposed *ADMIT* yields better performance.

JSMA [21] is an advanced attack compared to the FGSM and BIM, and is a more sophisticated attack. For JSMA, the perturbation is determined by the parameter η. We evaluate

Table 4.2: Performance (accuracy (%)) against blackbox attack on MNIST digits dataset

Variety of MNIST digits dataset		Blackbox attack			
Attack	Attack Parameter	No Defense	Random Nullification	Reconstructor	*ADMIT*
FGSM	0.3	27.77	10.02	90.74	91.90
BIM	0.3	23.13	9.87	91.53	92.87
JSMA	0.1	61.93	28.72	92.45	93.04
Carlini	5	22.01	19.71	68.95	73.68
DeepFool	0.1	41.95	23.05	96.39	96.95

Table 4.3: Performance (accuracy (%)) against whitebox attacks on MNIST digits dataset

	RNF Parameters		whitebox attack			
Attack	Mean	Standard Deviation	No Defense	Random Nullification	Reconstructor	*ADMIT*
FGSM	7	5	27.77	19.7	90.74	92.28
FGSM	7	10	27.77	23.98	90.74	92.59
FGSM	5	10	27.77	40.66	90.74	93.05
Carlini	7	5	22.01	23.03	68.95	76.15
Carlini	7	10	22.01	41.28	68.95	77.85
Carlini	5	10	22.01	38.55	68.95	76.01

the performance of the proposed *ADMIT*, reconstructor only and [22] (RNF only) techniques, whose performances are listed in Table 4.2. As seen that without defense, the adversaries lead to a performance of 61.93%. When employing defense like [22], the performance decreases to 28.72% in the case of blackbox attack. In such scenarios, as the adversaries are transferable, if the attack is generated on a different network, the performance of the reconstructor is lower compared to the proposed R^2AD. Similar behavior is observed with the DeepFool attack, as showcased in Table 4.2.

Lastly, we evaluate the performance against the CW attack, which is the advanced adversarial attack so far. The parameters utilized for providing perturbations(confidence) is set to 5. Table 4.2 shows the performance with CW attack utilizing L_2 norm. As one can observe that without deploying any defense, the performance is 22.01%. However, using the RNF layer can lower the performance. Similarly, the reconstructor shows a good performance; however, it is lower than the proposed *ADMIT*. Also, it needs to be noted that Table 4.2

shows the performance for the blackbox attack when using MNIST digits dataset.

As mentioned in the previous part, the experiments divide into two major parts. First of all, we evaluated the robustness of the proposed defense network against blackbox attacks. For assessing the effectiveness of this method, we craft MNIST digits data with five different attacks. Additionally, for FGSM and CW attacks, we try to measure the accuracy of the suggested method for whitebox attacks as well. In this scenario, the amount of perturbation is related to the mean and standard deviation of the RNF layer. In other words, the parameters of the RNF layer are known for the attacker. The parameters used for FGSM and CW in Table 4.3 are 0.3 and 5, respectively.

4.2.2 Evaluation with Fashion-MNIST Dataset

The noticeable efficiency of the proposed network on the MNIST digits dataset persuades us to investigate more on the correctness of *ADMIT*. To prove the effectiveness of *ADMIT*, we repeat the same experiment on the more sophisticated dataset named Fashion-MNIST. Fashion-MNIST consists of 10 labels like MNIST digits; but instead of hand-written digits, the labels are regarded to fashion like shoes and shirts. By comparing the accuracies for the network without defense and our proposed defense method (*ADMIT*) based on the information given in Table 4.2, FGSM and BIM are considered as the two top attacks with the most increase in their accuracies. Hence, we decided to use these two attacks with the Fashion-MNIST dataset. Table 4.4 contains the architectural details of the neural network used for the evaluation of *ADMIT*. Using the network presented in Table 4.4 will result in accuracy equal to 77.91% when evaluating Fashion-MNIST dataset. As mentioned in the previous sections, an autoencoder is the skeleton of the reconstructor, which is used in our proposed defense technique. The architectural details of the autoencoder used for

Table 4.4: Architecture of classifier and reconstructor for Fashion-MNIST dataset

classifier		reconstructor		
Conv+Relu	32×3×3	encoder	Conv+Relu	64×3×3
Max Pooling	2×2		Max Pooling	2×2
Dropout	0.25		Conv+Relu	32×3×3
Conv+Relu	64×3×3		Max Pooling	2×2
Max Pooling	2×2		Conv+Relu	16×3×3
Dropout	0.25		Max Pooling	2×2
Conv+Relu	128×3×3	decoder	Conv+Relu	16×3×3
Dropout	0.4		Up Sampling	2×2
Dense+Relu	128		Conv+Relu	32×3×3
Dropout	0.3		Up Sampling	2×2
Dense+softmax	10		Conv+Relu	64×3×3
			Up Sampling	2×2
			Conv+Relu	1×3×3

Fashion-MNIST dataset are also in Table 4.4.

The accuracy of our proposed network and other methods for the Fashion-MNIST dataset are presented in Table 9. The amount of perturbation added to the Fashion-MNIST dataset in both attacks is equal in comparison to these numbers for the MNIST digits dataset. In other words, the attack parameter for FGSM and BIM are considered as 0.3 and 0.3, respectively. For the FGSM attack, the accuracy of classification without any performance is 22.60%. In comparison to the MNIST digits, each image in the Fashion-MNIST dataset includes more pixels with valuable information. Therefore, nullifying some attacked pixels will decrease the amount of loss which is caused by the attack. Hence, using the random nullification layer will raise the accuracy around 6%. Similarly, we replicated the work presented in 3 where the reconstructor is used. Utilizing the reconstructor itself helps the network against the FGSM attack, and the performance reaches 54.42%. Lastly, the accuracy with applying the proposed *ADMIT*, will increase the accuracy to 59.12%.

Using the BIM attack on the Fashion-MNIST dataset will result in a similar pattern when compared to the FGSM attack. The accuracy of bare-network will grow when deploying the

Table 4.5: Performance (accuracy (%)) against blackbox attack on Fashion-MNIST dataset

Variety of Fashion-MNIST dataset		Blackbox attack			
Attack	Attack Parameter	No Defense	Random Nullification	Reconstructor	*ADMIT*
FGSM	0.3	22.60	28.20	54.42	59.12
BIM	0.3	15.91	24.63	55.62	60.48

random nullification layer. Among the four existing techniques, our results prove that the performance of our proposed *ADMIT*will defeat others in the Fashion-MNIST dataset as well.

The worth-mentioning point in all the experiments about the proposed reconstructor is that it has a very low rate of false negatives, which means that among discarded images (detected as adversarial data) only a few of them will be classified correctly by the network. Experiments show that in the lower amount of perturbation, this number is equal to zero. The more amount of perturbation leads to more number of false negatives, which will be as low as 15%.

Chapter 5

Conclusion

In this thesis, we proposed a method to defend against adversarial attacks that are intelligently crafted. To address this, we propose a 2 layer defense. Random Nullification will first try to minimize the effect of adversaries for the trained network. To address the impact of removed features by the random nullification layer, an autoencoder based reconstructor is introduced to rebuild the input features and ensure the rebuilt features follow the data distribution of normal samples. This reconstruction minimizes the impact of randomly nullified features. We advocate that the impressive defense against adversaries should not rely on any data from the process of generating adversarial data. In other words, it should be attack-independent. *ADMIT*is evaluated under whitebox and blackbox scenarios and have shown high robustness against adversaries and higher performance on normal samples. Experimental evaluation with different levels of noise is performed and compared with some of the recent defenses. It is seen that with the proposed *ADMIT*up to 80% higher performance is achieved compared to some of the recently proposed defenses.

Bibliography

[1] Marzieh Ashrafiamiri, Sai Manoj Pudukotai Dinakarrao, Amir Hosein Afandizadeh Zargari, Minjun Seo, Fadi Kurdahi, and Houman Homayoun. R2ad: Randomization and reconstructor-based adversarial defense for deep neural networks. In *2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD)*, pages 21–26. IEEE, 2020.

[2] M. Wess, S. M. P. Dinakarrao, and A. Jantsch. Weighted quantization-regularization in DNNs for weight memory minimization toward HW implementation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2929–2939, Nov 2018.

[3] H. Sayadi, N. Patel, P. D. Sai Manoj, A. Sasan, S. Rafatirad, and H. Homayoun. Ensemble learning for hardware-based malware detection: A comprehensive analysis and classification. In *ACM/EDAA/IEEE Design Automation Conference*, 2018.

[4] S. M. P. Dinakarrao and et al. Adversarial attack on microarchitectural events based malware detectors. In *Design Automation Conf.*, 2019.

[5] Rozhin Yasaei, Shih-Yuan Yu, Emad Kasaeyan Naeini, and Mohammad Abdullah Al Faruque. Gnn4ip: Graph neural network for hardware intellectual property piracy detection. *arXiv preprint arXiv:2107.09130*, 2021.

[6] Rozhin Yasaei, Shih-Yuan Yu, and Mohammad Abdullah Al Faruque. Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1504–1509. IEEE, 2021.

[7] Ali Tazarv and Marco Levorato. A deep learning approach to predict blood pressure from ppg signals. *arXiv preprint arXiv:2108.00099*, 2021.

[8] Seyed Amir Hossein Aqajari, Rui Cao, Amir Hosein Afandizadeh Zargari, and Amir M Rahmani. An end-to-end and accurate ppg-based respiratory rate estimation approach using cycle generative adversarial networks. *arXiv preprint arXiv:2105.00594*, 2021.

[9] Milad Asgari Mehrabadi, Seyed Amir Hossein Aqajari, Iman Azimi, Charles A Downs, Nikil Dutt, and Amir M Rahmani. Detection of covid-19 using heart rate and blood pressure: Lessons learned from patients with ards. *arXiv preprint arXiv:2011.10470*, 2020.

[10] Ali Tazarv, Sina Labbaf, Stephanie M Reich, Nikil Dutt, Amir M Rahmani, and Marco Levorato. Personalized stress monitoring using wearable sensors in everyday settings. *arXiv preprint arXiv:2108.00144*, 2021.

[11] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75, Aug 2018.

[12] R. S. Gutzwiller and J. Reeder. Human interactive machine learning for trust in teams of autonomous robots. In *IEEE Conf. on Cognitive and Computational Aspects of Situation Management*, 2017.

[13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

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

[15] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy (Euro S&P)*, 2016.

[16] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. In *International Conference on Learning Representations (ICLR)*, 2017.

[17] S. M. P. Dinakarrao and et al. Enhancing adversarial training towards robust machine learners and its analysis. In *Int. Conf. on Computer-Aided Design*, 2018.

[18] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: increasing local stability of neural nets through robust optimization. *ArXiv e-prints*, 2015.

[19] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *Int. Conf. on Learning Representations*, 2017.

[20] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick D. McDaniel. On the (statistical) detection of adversarial examples. *CoRR*, abs/1702.06280, 2017.

[21] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[22] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G. Ororbia, II, Xinyu Xing, Xue Liu, and C. Lee Giles. Adversary resistant deep neural networks with an application to malware detection. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

[23] J. Buckman, A. Roy, and I. Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *Int. Conf. on Learning Representations*, 2018.

[24] Amir Hosein Afandizadeh Zargari, Manik Dautta, Marzieh Ashrafiamiri, Minjun Seo, Peter Tseng, and Fadi Kurdahi. Newertrack: Ml-based accurate tracking of in-mouth nutrient sensors position using spectrum-wide information. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3833–3841, 2020.

[25] Rozhin Yasaei, Felix Hernandez, and Mohammad Abdullah Al Faruque. Iot-cad: context-aware adaptive anomaly detection in iot systems through sensor association. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.

[26] Amir Hosein Afandizadeh Zargari, Seyed Amir Hossein Aqajari, Hadi Khodabandeh, Amir M Rahmani, and Fadi Kurdahi. An accurate non-accelerometer-based ppg motion artifact removal technique using cyclegan. *arXiv preprint arXiv:2106.11512*, 2021.

[27] Seyed Amir Hossein Aqajari, Rui Cao, Emad Kasaeyan Naeini, Michael-David Calderon, Kai Zheng, Nikil Dutt, Pasi Liljeberg, Sanna Salanterä, Ariana M Nelson, and Amir M Rahmani. Pain assessment tool with electrodermal activity for postoperative patients: Method validation study. *JMIR mHealth and uHealth*, 9(5):e25258, 2021.

[28] Seyed Amir Hossein Aqajari, Emad Kasaeyan Naeini, Milad Asgari Mehrabadi, Sina Labbaf, Nikil Dutt, and Amir M Rahmani. pyeda: An open-source python toolkit for pre-processing and feature extraction of electrodermal activity. *Procedia Computer Science*, 184:99–106, 2021.

[29] Yann LeCun and et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[30] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[31] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP)*, 2017.

[32] A. Kurakin, I.J. Goodfellow, and S. Bengio. Adversarial examples in the physical world. In *International Conference on Learning Representations*, 2017.

[33] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599, 2015.

[34] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *ArXiv e-prints*, 2015.

[35] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *ACM Conf. on Computer and Communications Security*, 2017.

[36] John Bradshaw, Alexander G de G Matthews, and Zoubin Ghahramani. Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks. *arXiv preprint arXiv:1707.02476*, 2017.

[37] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[38] Martín Abadi and et al. Tensorflow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[39] Nicolas Papernot and et al. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.