DEFENDING AGAINST ADVERSARIAL ATTACKS WITH DENOISING AUTOENCODERS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Rehana Mahfuz

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

May 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

ACKNOWLEDGMENTS

I express my sincere gratitude to my advisor, Prof. Aly El Gamal, for mentoring my project, for giving me a platform to make mistakes and learn, and for providing constructive feedback. I am grateful not only for the direct supervision of my research project he provided, but also for the advice and support he has provided throughout the pursuit of my Master's degree. Next, I would also like to thank other members of my advisory committee, Prof. Milind Kulkarni and Prof. Maggie Zhu, for providing guidance and helpful advice in my transition from a Bachelor's degree to a Master's degree, and for always being available.

I also take this opportunity to acknowledge my labmate Rajeev Sahay for the fruitful discussions we had regarding subject matter of our research. Further, I would like to thank my labmate and officemate Teng-Hui Huang for the pleasure of the discussions we had about graduate school and research. Finally, I would like to thank family and friends that have supported and encouraged me throughout.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SYMBOLS

x	input data sample
y	label corresponding to input data sample x
$\|C\|$	number of classes in the classification task
θ	weights of a trained machine learning model
$f_k(x)$	classification score of data point x for class k
$Z_k(x)$	logit corresponding to class k in classification of data sample x
$C^*(x)$	classification label of data sample x assigned by unattacked machine learning model
$J(x, y, \theta)$	loss function when classifying data sample x with true label y using a model with parameters θ
$D(x)$	output of a denoising autoencoder when data sample x is forward propagated through it
δ	perturbation added to an input sample x in an effort to make it adversarial

ABBREVIATIONS

CW	Carlini-Wagner
DF	Deepfool
FGS	Fast Gradient Sign
DAE	Denoising Autoencoder
Dim Red	Dimensionality Reduction

ABSTRACT

Mahfuz, Rehana M.S., Purdue University, May 2020. Defending Against Adversarial Attacks with Denoising Autoencoders. Major Professor: Aly El Gamal.

Gradient-based adversarial attacks on neural networks threaten extremely critical applications such as medical diagnosis and biometric authentication. These attacks use the gradient of the neural network to craft imperceptible perturbations to be added to the test data, in an attempt to decrease the accuracy of the network. We propose a defense to combat such attacks, which can be modified to reduce the training time of the network by as much as 71%, and can be further modified to reduce the training time of the defense by as much as 19%. Further, we address the threat of uncertain behavior on the part of the attacker, a threat previously overlooked in the literature that considers mostly white box scenarios. To combat uncertainty on the attacker's part, we train our defense with an ensemble of attacks, each generated with a different attack algorithm, and using gradients of distinct architecture types. Finally, we discuss how we can prevent the attacker from breaking the defense by estimating the gradient of the defense transformation.

1. INTRODUCTION

The ability of neural networks to learn from exposure to training data closely mimics the ability of humans to learn from experience. Just as humans are employed for tasks that have no easy solutions and therefore require critical thinking and deliberation, neural networks are now being employed for tasks which have no closed-form mathematical solution, and hence require pattern recognition. Any system that involves subjectivity in making decisions is likely to be fooled by certain inputs. Just as humans are fooled by optical illusions, neural networks have also been found to be fooled by adversarial examples [1].

1.1 What are adversarial attacks?

Adversarial attacks are ways to perturb data that is input into a machine learning model, with the aim of deteriorating the performance of the model. We refer to this machine learning model as the *victim model* or the *victim network*. Specifically, we consider neural networks used for classification tasks as our machine learning model. The attacker's goal is to make the output of the machine learning model incorrect. The attacker can achieve this goal by either influencing the training process of the victim model, or by corrupting the test data. The former is known as a *poisoning attack*, where the attacker injects malicious samples into the training dataset of the victim model, so that it is trained to misclassify. The latter is known as an *evasion attack*, which involves perturbing the test data, often imperceptibly. The occurrence of poisoning attacks is subject to the very strong assumption that the attacker will be able to influence the training process. Hence, in our work, we only consider the more realistic evasion attacks which only require perturbation of the test samples.

Formally, an evasion adversarial attack on a machine learning model $f(x, y, \theta)$ is constructed by finding the minimum perturbation δ to be added to input x such that the classification decision of the model changes. The attacked data sample is $x' = x + \delta$, where

$$\delta = \min_{\beta} ||\beta||_a \; such\, that \; f(x + \beta) \neq f(x) \tag{1.1}$$

where a is a norm usually chosen to be 0, 2 or ∞. Minimizing the l_0 norm would minimize the number of features that are modified. Minimizing the l_2 norm would minimize the Euclidean distance between the attacked image and original image, which is sometimes a good measure of perceptibility. Minimizing the l_∞ norm minimizes the magnitude of maximum perturbation that can be applied to any feature. In our experiments, we use the version of the attack that minimizes the l_2 norm.

Intuitively, adversarial attacks can be viewed as data samples that lie very close to the classification decision boundary. Since the decision boundary of every neural network is subject to multiple random factors such as random initialization of weights, the decision boundary of a neural network is never perfect. Attackers take advantage of such imperfection and move data points belonging to one class across the decision boundary so that the neural network classifies it as belonging to a different class. This is the methodology of computing adversarial attacks using the Deepfool method, which is discussed later. This view is reinforced by [2], which suggests that adversarial images occupy dense regions in the pixel space, instead of existing as isolated points in that space. More generally, this would mean that adversarial examples occupy dense regions in the feature space.

1.2 Contributions of this work

We see that adversarial attacks have the ability to reduce the accuracies of neural networks which are relied on for extremely critical applications such as medical diagnosis [3–5] and biometric authentication [6–8]. Hence the need of the hour is to make these neural networks robust to adversarial perturbations. We propose a method to recover the accuracy of an attacked network by leveraging the power of Denoising Autoencoders (DAEs), and combining that with dimensionality reduction. Further, we also customize our defense to maintain robustness when there is uncertainty about the attacker's choices regarding construction of the attack. Our defense has the following merits:

1. **Effective against an unknown attack type generated using gradients of an unknown network**

   Existing defense mechanisms are often studied only in scenarios where the attacker is likely to use the strongest attack possible against a given network. However, the attacker may not always have enough information or computational resources to craft such an optimal attack. It is therefore important to find effective defenses when the attack may be crafted differently from the optimal. We take a step in that direction by evaluating the effectiveness of defenses trained to denoise multiple attacks, which are generated with distinct choices of the parameters which the attacker can vary while generating the attack. Variation of these parameters by the attacker leaves the defender with uncertainty. We empirically determine the smartest way to handle such uncertainty.

2. **Customizable for enhanced computational efficiency**

   Our proposed defense can be modified to reduce training time of the classifier by as much as 71%. This may make tasks that were previously computationally infeasible more feasible. Further, it is also possible to modify our defense to reduce the training time of the defense itself by as much as 19%, as explained in Chapter 4. This comes with a small tradeoff in the performance of the defense.

3. **Suitable for any data type, not just images**

   Several existing defenses such as JPEG compression [9, 10], color bit depth reduction [11], and random resizing and random padding [12] are applicable only to defend image data against adversarial attacks. Our proposed defense is suitable for many data types involving many different applications such as income prediction from socio-economic details, human activity recognition from sensor data, medical diagnosis from patient history, identifying type of glass from their oxide content for criminological investigation, etc.

4. **May be cascaded with other detection/defense mechanisms** For any defense, there is generally a robustness-accuracy tradeoff [13], which means that using the defense is likely to compromise the performance of the machine learning model. Hence it is wise to use the defense only in the presence of a non-trivially strong attack. Not all defense mechanisms offer the ability to selectively defend test samples. Our defense affords us the ability to apply the denoising mechanism selectively to samples that are detected as being adversarial, and letting the remaining samples be processed in a standard manner. Thus our denoising defense may be used in series with detection methods such as exploiting convolution filter statistics [14] and SafetyNet [15].

The remaining document is structured as follows. In the next chapter, we provide an overview of the related work, which includes adversarial attacks and the defenses that have been proposed to counter them. In Chapter 3, we propose a defense built on the DAE, which also reduces training time of the classifier. Chapter 4 discusses modification of this defense to reduce the training time of the defense itself. In Chapter 5, we investigate the performance of our defense in scenarios where the attacker behaves unexpectedly, by customizing our defense to take into consideration different choices made by the attacker in generating the attack. Finally, Chapter 6 wraps up our discussion of this proposed defense against adversarial attacks.

2. BACKGROUND

2.1 Adversarial Attacks

2.1.1 Fast Gradient Sign Attack

The Fast Gradient Sign (FGS) attack [16] views the input x as a space where small changes can be made with the objective of minimizing the cost function $J(\theta, x, y)$. Hence it adds a perturbation δ in the direction of the sign of the gradient of the cost function with respect to the input. While imposing an upper bound of ϵ on the l_∞ norm of the perturbation, ϵ is chosen to be the magnitude of this perturbation, as shown in Equation 2.1.

$$\delta = \epsilon sign(\Delta_x J(\theta, x, y)) \tag{2.1}$$

To generate an attack bounded by an l_2 norm of ϵ, this gradient is divided by its l_2 norm, as shown in Equation 2.2.

$$\delta = \epsilon \frac{\Delta_x J(\theta, x, y)}{||\Delta_x J(\theta, x, y)||_2} \tag{2.2}$$

Apart from being fast because of requiring only one gradient computation, another advantage of this method is the attack is general enough to transfer to other networks trained to perform the same task. This is because it takes a step of a certain size in the direction of the sign of the gradient, which may be larger than the minimum step required for that particular classifier.

2.1.2 Deepfool Attack

The Deepfool (DF) attack [17] attempts to move the data sample across the boundary of the closest separating hyperplane. This is performed for a certain number of iterations (generally 50), or until the classifier is found to misclassify the perturbed data. At every iteration i, by assuming the decision boundaries to be linear at the point closest to the data point x_i, a region $\tilde{P}_i$ is approximated such that x_i would be correctly classified in that region:

$$\tilde{P}_i = \bigcap_{k=1}^{|C|} \{x : f_k(x_i) + \nabla f_k(x_i)^T x \leq f_{C^*(x_0)}(x_i) + \nabla f_{C^*(x_0)}(x_i)^T x\} \tag{2.3}$$

At iteration i, the perturbation δ_i is calculated as:

$$\delta_i = \frac{|f_l(x_i) - f_{C^*(x_0)}(x_i)|}{||\nabla f_l(x_i) - \nabla f_{C^*(x_0)}(x_i)||_2^2} (\nabla f_l(x_i) - \nabla f_{C^*(x_0)}(x_i)) \tag{2.4}$$

where l is the index of the class closest to x_0, which is found as:

$$l = argmin_{k \neq C^*(x_0)} \frac{|f_k(x_i) - f_{C^*(x_0)}(x_i)|}{||\nabla f_k(x_i) - \nabla f_{C^*(x_0)}(x_i)||_2^2} \tag{2.5}$$

This is a good strategy to use if the attacker has access to the victim classifier's exact weights, since this finds a very small perturbation that is effective in causing the classifier to misclassify. However, this attack does not work very well if deployed on a different classifier, since it perturbs the data only as much as required for that specific classifier whose gradients are used to craft the attack. Thus the Deepfool attack is not transferable.

2.1.3 Carlini-Wagner Attack

The Carlini-Wagner (CW) attack [18] solves a carefully constructed optimization problem using gradient descent from multiple random starting points. The first term of the objective function to be minimized is the squared l_2 norm of the perturbation. The second term is a term that is 0 if and only if misclassification occurs. In other words, it is a constraint that is introduced as a term in the objective. Out of many candidates for such a term, the authors empirically determined that $max(Z_{C^*(x)}(x + \delta) - \max_{i \neq C^*(x)}\{Z_i(x + \delta)\}, 0)$ is the most effective choice. This second term is scaled by a constant c to adjust the relative importance of the terms. Such a constant c is determined by using binary search. Formally, the optimal perturbation δ is found as:

$$\min_{\delta} ||\delta||^2 + c.max(Z_{C^*(x)}(x + \delta) - \max_{i \neq C^*(x)} \{Z_i(x + \delta)\}, 0) \tag{2.6}$$

Moreover, for image data, since the perturbed data sample $x+\delta$ has to lie in a certain range to be interpreted as a pixel, a change of variable is applied to δ to ensure that it lies between 0 and 1:

$$\delta = \sigma(2w) - x \tag{2.7}$$

The success of this attack can be attributed to three factors: i) the carefully chosen form of the constraint in the second term, ii) the carefully chosen constant c that scales the second term, and iii) the use of multiple random starting points for gradient descent.

2.2 Existing Defenses

2.2.1 Network Distillation

Neural network distillation [19] involves training the victim network with soft or continuous labels, instead of hard or discrete labels. These soft labels are the prediction probabilities output by a neural network trained to perform the same task. To make these probabilities more continuous and less discrete, the softmax layer is modified as follows, where the temperature T is set to a value greater than 1.

$$f_k(x) = \frac{e^{\frac{z_k(x)}{T}}}{\sum_{l=1}^{|C|} e^{\frac{z_l(x)}{T}}} \tag{2.8}$$

A high temperature is used to train the network used to generate soft labels, and also to train the network which is going to be used as the actual classifier. However, the temperature is set to 1 while testing. This defense has been overcome by the CW attack [18].

2.2.2 Adversarial Training

When a network is trained to correctly classify adversarial examples in addition to classifying clean samples, that process is known as adversarial training. Such a process is very computationally intensive since it requires more training. One variant is cascade adversarial training [20], where the victim network is trained with adversarial examples generated using a different network, in addition to adversarial examples generated using the same network. [21] trained the generator of a Generative Adversarial Network (GAN) to generate adversarial examples while the victim classifier was being trained. The victim classifier was trained to correctly classify both clean and perturbed samples, as the perturbed samples were being generated by the generator which was also being trained.

2.2.3 Data Preprocessing

Several approaches have been proposed to apply transformations to the input before feeding it into the victim network.

Compression

Reducing the dimensionality of data generates a representation of the data using fewer dimensions which are able to capture only the large variation. Hence this has been used as a common approach to defend against adversarial attacks, in the hope that it will exclude the adversarial perturbation, since it is small. [22] proposes using Principal Components Analysis (PCA). [23] uses an autoencoder to reduce the dimensionality of the input. An autoencoder is a neural network whose successive layers decrease in the number of neurons before increasing again to the original. It is trained to reconstruct the original input, which necessitates trimming down the data to a small number of dimensions to propagate through the smallest layer. The output of this smallest layer is the representation of the input data in a reduced number of dimensions. Another method specifically used for images is JPEG compression [9,10]. Comparing the effectiveness of JPEG compression with PCA, low-pass filtering, low resolution wavelet approximation and soft thresholding, [24] found JPEG compression to be most effective as a defense. As with other preprocessing defenses, this can be overcome by the attacker if they use a differentiable approximation of the JPEG transformation, as shown by [25]. A general problem with input compression is that high compression causes a loss in the classification accuracy.

Denoising Autoencoder

Attempts have been made to remove the perturbation from the test input. One way is to use a Denoising Autoencoder (DAE). This has an autoencoder structure similar to that of an autoencoder used for dimensionality reduction. However, the training method is different. While training, a DAE is presented with both clean data and corrupted data as the input, and is trained to produce only clean data at its output. This output is then fed into the classifier. MagNet [26] uses one of many trained DAEs as a *reformer*, on examples it has detected as adversarial. The *detector* is an autoencoder that has learnt the manifold of clean data, and declares an example as being adversarial if it is more than a certain distance away from this manifold. However, MagNet only trains its DAEs on data corrupted by adding Gaussian noise, and not on data corrupted by adding adversarial perturbations. Thus its detector and reformer have only been tested with very small perturbations, with an l_2 norm of upto 1.0. Perturbations with l_2 norms larger than 1.0 continue to be imperceptible, as shown in Figures 3.2 and 5.1. Another use of a DAE has been seen in DUNET [27], which trained a convolutional DAE to learn the noise which can then be subtracted from the corrupted examples. This works well for color images, for which a simple DAE is insufficient.

Other methods

For image data, feature squeezing [11] may be performed by reducing a pixel's color bit depth, or by performing spatial smoothing. This was used for only detection of adversarial examples. Randomly resizing the test image and adding random padding also reduces the success rate of the attacker [12]. **??** proposed two modifications. The first modification is to use Bounded ReLU activation function, which is $BReLU(x) =$

$min(max(x, 0), 1)$, instead of the ReLU [28] activation function, which is $ReLU(x) = max(x, 0)$. The second modification is to train the network with Gaussian noise added to the training data. Both of these defenses have been broken in [29].

[30] trained a GAN such that the generator G generates clean images, while the discriminator tries to correctly judge if the image is real or is artificially generated by the generator. During test time, an estimation $G(z^*)$ of the possibly perturbed test image x is projected onto the range of the generator, using L steps of gradient descent, which is attempted for R starting points of gradient descent. Gradient descent is performed L times to minimize $||G(z^{(i)}) - x||_2^2$ as a function of $z^{(i)}$, where $i = 1, ..., R$. The notation $z_j^{(i)}$ is used to for the value of $z^{(i)}$ at the j^{th} iteration of gradient descent. z^* is chosen as ${}_{z\in\{z_L^{(1)},...,z_L^{(R)}\}}||G(z) - x||_2^2$. This estimation $G(z^*)$ is then input into the neural network. Another work, APE-GAN [31], attempted to use the generator of a GAN to remove perturbation by accepting perturbed input and generating clean output. APE-GAN was shown to be ineffective in defending against the Carlini-Wagner attack [29].

2.3 Overcoming defenses

It has been claimed that many of these defense strategies are effective because they obfuscate the gradient of the network that is being defended [32]. The first type of obfuscated gradient is a *shattered gradient*, which happens when a gradient is non-differentiable, nonexistent, incorrect, or causes numeric instability, such as that seen in thermometer encoding [33], and in input transformations such as JPEG compression for images [34]. This can be overcome by estimating a differential approximation of the non-differentiable layer, in a method termed as Backward Pass Differentiable Approximation (BPDA). The second type of obfuscated gradient is a *stochastic gradient*, which happens in defenses that use randomization either in the input transformation or in the network itself, which is why using gradients from a single instance of that random defense may be insufficient to generate an attack that is likely to be effective in every instance of that random defense, such as that seen in Stochastic Activation Pruning (SAP) [35] and in Mitigating Through Randomization [36]. This is overcome by applying Expectation over Tranformation (EOT) to compute the gradient of the expected transformation to the input. The third type of obfuscated gradient is an *exploding* or a *vanishing gradient*, which may be caused by multiple iterations of evaluation by a neural network, as seen in PixelDefend [37] and in Defense-GAN [30]. This can be overcome by applying a change-of-variable to the input x such that the function performing the optimization loop that leads to exploding/vanishing gradients can be approximated by this new variable.

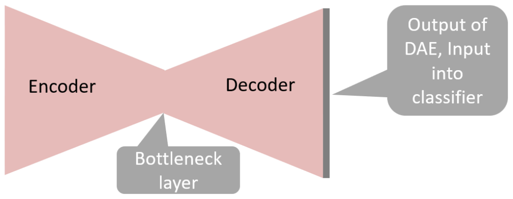
From this point of view, most defenses that process the input before feeding it into the classifier can be overcome. All that needs to be done is estimation of the gradient of the input transformation. Even if the input transformation is not differentiable, a differentiable approximation of that transformation can be approximated for the purpose of generating the attack, as is done to break JPEG compression as a defense [25]. The DAE defense is viewed as using a shattered gradient, and can be overcome by the attacker using BPDA, as illustrated in [29]. This is done by modifying the CW attack such that the second term of the objective function now looks at the decision of the classifier on the perturbed input x' which has been transformed using the DAE, as shown in Equation 2.9.

$$minimize||\delta||^2 + c.max(Z_{C^*(x)}(D(x')) - \max_{i \neq C^*(x)}\{Z_i(D(x'))\}, 0) \tag{2.9}$$

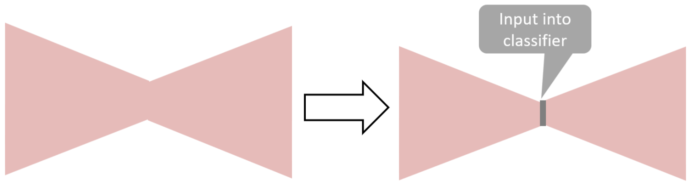
These methods to estimate the gradient work when the attacker knows exactly what the defense strategy is. However, with the availability of multiple defense strategies which the defender can choose from at run time, the attacker can never have certainty about which defense strategy is exactly being deployed. In such a case, the attacker cannot find an estimation of the gradient of the defense that satisfactorily approximates each of the possible defenses. Hence, this method of overcoming the defense is unlikely to work if the exact defense is randomly chosen at run time.

3. CASCADED DEFENSE

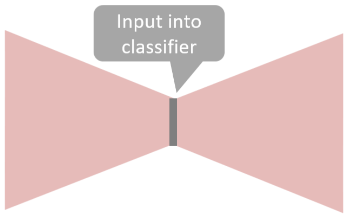
3.1 Motivation



(a) DAE defense.



(b) Cascaded defense.



(c) Hidden Layer defense.

Fig. 3.1. Our three considered defenses.

Given that an evasion adversarial attack perturbs test samples before being fed into the machine learning classifier, our first idea to remedy this was to attempt to remove this noise as part of the preprocessing step. Systems which use machine learning models generally involve a step to pre-process the data and transform it into the feature space that the machine learning models expect. For example, image data is often centered, cropped and resized to put the subject of the image in the center. Natural language data is converted into word embeddings which are fed into the machine learning classifier. We envision our add-on defense to be part of such a preprocessing module. We chose to use a Denoising Autoencoder (DAE) to remove the noise, since this method is agnostic to the nature of data. It can be used for medical diagnosis data, student loan relational database etc. Further, we decided to test the effect of reducing the dimensionality of this denoised data. This step also works for any data type. This is noteworthy because defenses such as feature squeezing and JPEG compression are only effective for image data.

3.2 Procedure

An autoencoder is a neural network which has two parts: an encoder and a decoder. The encoder has multiple layers of decreasing sizes, starting with the first layer whose size is the same as the dimensionality of input data. Here, the size of a layer is the number of neurons in the layer. The decoder is the name given to the remaining autoencoder which has layers of increasing sizes. Generally, the size of the $s-i+1^{th}$ layer in the decoder is the same as the size of the i^{th} layer in the encoder, where s is the total number of layers in the autoencoder. The last layer of the encoder, which is also the smallest layer, is called the bottleneck layer, because it limits how much information can flow through the network. All layers are fully connected.

A Denoising Autoencoder (DAE) is an autoencoder that we use to remove perturbation from data, as shown in Figure 3.1(a). While training it, we present it with some unperturbed input samples as well as some perturbed samples. The target to be learned is always the corresponding clean sample. Since the bottleneck layer only allows a limited amount of information to flow through the DAE, the network has to choose which information to discard. By training it to learn a clean representation of the data, we are forcing the DAE to learn to discard the perturbation, and only allow the clean data to flow through the network till the output. We also used an autoencoder to reduce the dimensionality of data. To do this, while training, we present the autoencoder with the same data as both the input and the target output. Since the network is trained to reconstruct the same data, the encoder simply gets trained such that the output of the bottleneck layer provides a compressed representation of the input data.

To prepare the defense, these two autoencoders are trained separately. Also, we train a victim classifier with a modified architecture, where the size of the input layer is equal to the dimensionality of the compressed data. While deploying the defense, the data is denoised using the DAE, after which it is compressed. Then this data is sent as input into the classifier, as shown in Figure 3.1(b). Since this method involves cascading the dimensionality reduction module with the denoising module, we call it a Cascaded defense.

3.2.1 Implementation Details

We used two datasets to test our defense. MNIST-Digit [38] is a dataset of 70,000 28x28 pixel grayscale images of handwritten digits. The classification task is to classify each image into one of ten labels, where each label corresponds to a digit in the decimal system. While MNIST-Digit is a standard dataset used to evaluate the performance of classification tasks, it may be insuffcient because of being too simple. Hence we also chose to use the Fashion-MNIST dataset [39]. Here, the classification task is to classify each image into one of ten fashion items such as *trouser*, *dress*, *pullover* and *sneaker*. Both datasets have 60,000 training images and 10,000 test images.

Since both Fully Connected (FC) neural networks as well as Convolutional Neural Networks (CNN) are able to solve these classification tasks, we use both types of architectures. The victim FC classifier is a neural network which has 784 neurons in the first layer, 100 neurons in the second layer and third layers each, followed by 10 neurons in the fourth layer. We describe this architecture as FC-784-100-100-10. More generally, the architecture FC-$n_1 - n2 - ... - n_k$ describes a neural network which has n_1 neurons in its first layer, n_2 neurons in its second layer, and so on, till the k^{th} layer, which has n_k neurons. In the real world, it seems unlikely that the attacker will use exactly the same architecture as the victim classifier's to generate the attack. To begin with, it may be difficult for the attacker to gather information about the exact victim classifier. Even if the attacker has that information, they

Table 3.1.
MNIST adversary's CNN architecture.

Conv 3x3x32, ReLU
Conv 3x3x64, ReLU
Max Pool 2x2
Dropout (rate = 0.25)
FC (128 neurons), ReLU
Dropout (rate = 0.5)
Softmax (10 classes)

may still choose to use a different architecture because of various reasons such as avoiding training a new classifier for the purpose of generating the attack. Hence we use a different neural network to generate the attack. The adversary's FC network has architecture FC-784-200-100-100-10. In both classifiers, all layers have Rectified Linear Unit (ReLU) activation except the last layer, which has softmax activation.

Both classifiers are trained for 100 epochs with a batch size of 200, with categorical crossentropy loss and Adam optimizer, with a learning rate of 0.001. For the MNIST-Digit dataset, the victim classifier achieves an accuracy of 98.11%, and the adversary's classifier achieves an accuracy of 98.38%. For the Fashion-MNIST dataset, the victim classifier achieves an accuracy of 88.56%, and the adversary's classifier achieves an accuracy of 88.42%.

The adversary's CNN has an architecture as shown in Table 3.1, and achieves an accuracy of 98.61% for the MNIST-Digit dataset and 93.32% for the Fashion-MNIST dataset. The victim CNN has a similar architecture, except that it has only two convolutional layers followed by a softmax layer. It achieves an accuracy of 98.66% for the MNIST-Digit dataset, and 91.06% for the Fashion-MNIST dataset. Both of the CNNs were trained for 20 epochs with a batch size of 200, with categorical crossentropy loss and Adam optimizer, which used a learning rate of 0.001.

Note that these classifier architectures were slightly modified to adjust the size of the input layer to match the size of the compressed data. In this case, the classifier was trained to classify data that has been compressed by the autoencoder.

The DAE has architecture FC-784-256-128-81-128-256-784. None of the layers have any activation except the final layer, which has sigmoid activation. This DAE was trained for 150 epochs with a batch size of 200, with Mean Squared Error (MSE) loss and Adam optimizer which uses a learning rate of 0.001.

Each DAE is trained with perturbations generated using only one attack algorithm, and using gradients of a specific architecture type. Here, architecture type refers to either FC or CNN. While testing the performance of the defense, we specifically test its effectiveness in mitigating an attack generated using the same attack algorithm and using gradients of the same architecture type. Exploring the effectiveness of a defense trained using perturbations generated by a particular algorithm and using gradients of a particular architecture type in mitigating different types of attacks is very interesting, and we explore that in Chapter 5.

For each dataset, we trained one autoencoder to reduce the dimensionality of data. This autoencoder has architecture FC-784-81-784. It reduces the data to 81 dimensions from 784 dimensions, which can also be interpreted as 9x9 dimensions from 28x28 dimensions. Hence the size of the input layer of the victim FC network is 81, and the size of the input layer of the victim CNN is 9x9. This autoencoder was trained with the Adam optimizer with a learning rate of 0.001, MSE loss, 100 epochs and with a batch size of 500. It had ReLU activation in all layers except sigmoid activation in the last layer.

All attacks were generated using the TensorFlow Cleverhans library [40] and were untargeted attacks. The hyperparameters of all three simulated attacks were adjusted such that the attacks are reasonably imperceptible as well as effective at the same time. The l_2 norm of the FGS attack was set to 2.5 when using gradients of an FC network, and to 1.5 when using gradients of a CNN. Such attacked images are shown in Figure 3.2 for the MNIST-Digit dataset. For the Deepfool attack, there are no

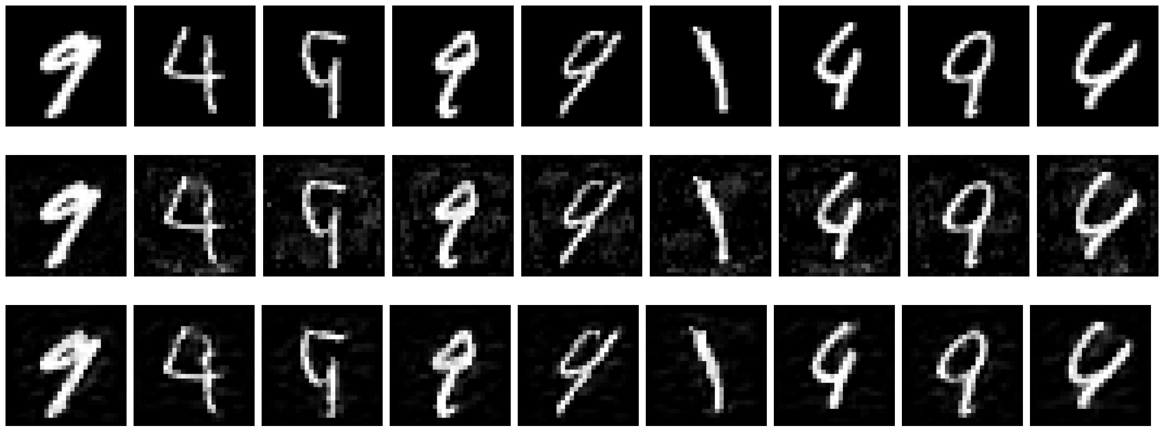
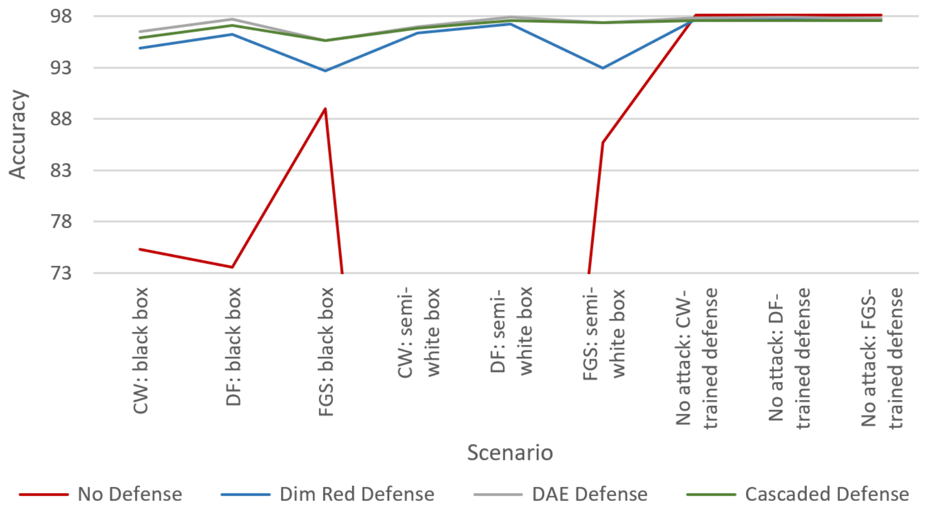


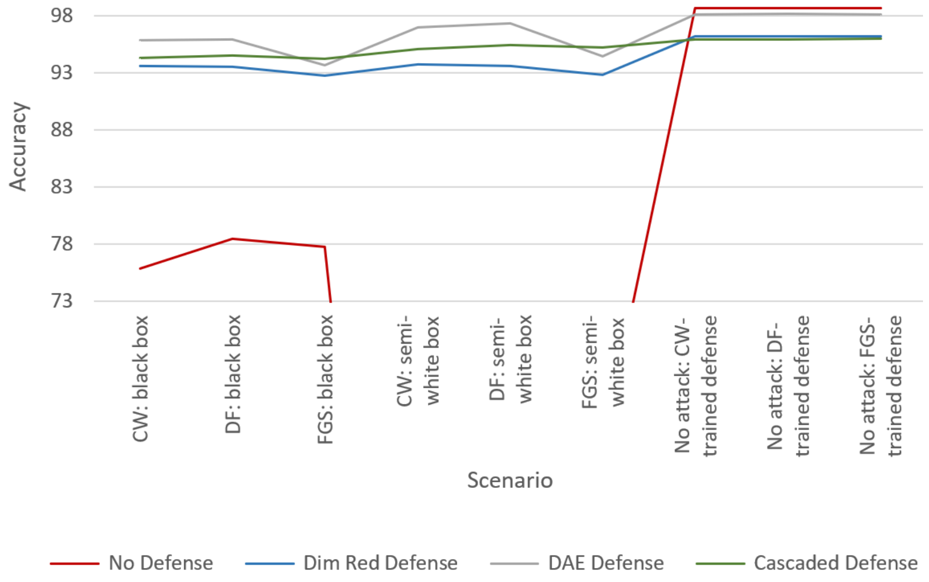
Fig. 3.2. MNIST-Digit images attacked with the FGS algorithm. The upper row has unperturbed images. The middle row has images perturbed using gradients of an FC classifier with l_2 norm 2.5. The bottom row has images perturbed using gradients of a CNN classifier with l_2 norm 1.5.

major parameters to tune except the maximum number of iterations, which was set as 50. The CW attack was generated with 4 binary search steps, a maximum of 60 iterations, a learning rate of 0.1, a batch size of 10, an initial constant of 1.0, and the abort_early parameter was set to True. All other parameters were left unchanged at their default values.

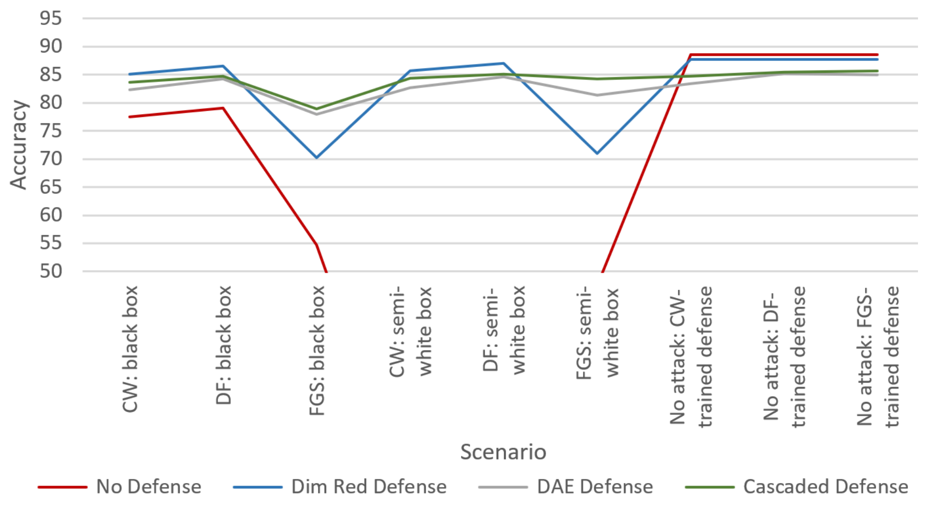
3.3 Results



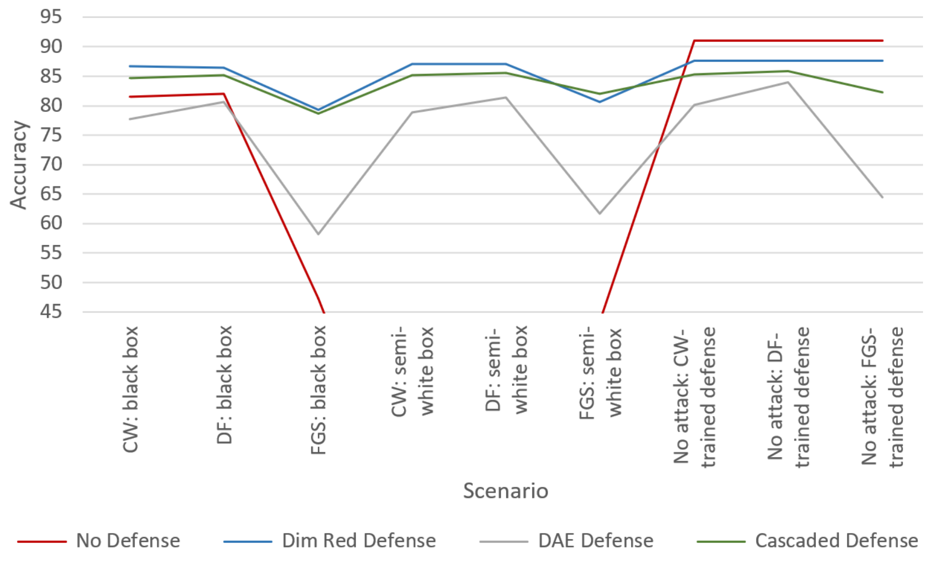
(a) Accuracies when the MNIST-Digit task uses an FC network. Post-attack accuracy in the semi-white box case for (i) CW: 1.24% (ii) DF:1.25%



(b) Accuracies when the MNIST-Digit task uses a CNN. Post-attack accuracy in the semi-white box case for (i) CW: 1.08% (ii) DF:1.09% (iii) FGS: 60.24%



(c) Accuracies when the Fashion-MNIST task uses an FC network. Post-attack accuracy in the semi-white box case for (i) CW: 8.09% (ii) DF: 7.98% (iii) FGS: 47.48%



(d) Accuracies when the Fashion-MNIST task uses a CNN. Post-attack accuracy in the semi-white box case for (i) CW: 8.09% (ii) DF: 7.98% (iii) FGS: 47.48%

Fig. 3.3. Performance of the Cascaded defense.

We evaluate the performances of the three defenses: the Dimensionality Reduction defense, the DAE defense and the Cascaded defense in both black box and semi-white box settings. In the black box setting, the adversary generates an attack using their own classifier, which is different from the victim classifier. In the semi-white box setting, the adversary uses gradients of the trained victim model to generate the attack. Note that having access to the trained victim model is different from merely knowing the architecture of the victim model. This is because of random initialization of weights in the model. If the attacker uses the exact trained model, the attack generated will be much stronger than an attack generated using a model with the same architecture which was trained using the same data and parameters (learning rate, optimizer, number of epochs, batch size etc.), but with differently initialized weights. To be more conservative in the evaluation of our defense, we used the stronger attack, which is generated using the trained victim model. A gray box setting would be when the attacker also knows about our defense strategy of using a DAE. A white box setting would be when the attacker has access to our exact trained DAE. However, since our defense is part of the data preprocessing pipeline, hijacking of the data preprocessing pipeline by the attacker is a completely different problem, which we do not consider in our work. However, in Chapter 6, we do discuss a general strategy to prevent the attacker from overcoming defenses in a white box scenario, which can be applied to our defense as well. Further, we discuss possibilities of the gray box and uncertain black box settings in Section 2.3

Figure 3.3 is divided into four parts, where each part shows the performance of our defenses for a particular dataset when a specific architecture type is used to perform the task. In the legend of Figure 3.3, 'Dim Red' refers to 'Dimensionality Reduction'. In the semi-white box case, the accuracies without defense are low enough to be outside the scope of representation of the plot, which is why we have mentioned these accuracies in the subfigure captions when they cannot be shown in the subfigure. To defend against an attack generated using gradients of a specific architecture type and using a specific algorithm, we used a DAE that was trained to denoise an attack generated using gradients of the same architecture type and using the same algorithm. In practice, it may be hard to predict which architecture type and which attack algorithm the adversary will choose, which is why there is a need to develop a more universal defense. Chapter 5 takes a step towards investigating that.

When the MNIST-Digit data is attacked by an FC network, the DAE and Cascaded defenses perform similarly, while the Dimensionality Reduction performs a little worse. When the MNIST-Digit data is attacked by a CNN, the Cascaded defense performs most consistently, while the Dimensionality Reduction defense performs a little worse. The DAE defense yields an accuracy higher than the other defenses for CW and DF attacks, but this accuracy reduces for an FGS attack. When the Fashion-MNIST data is attacked by an FC network, the Cascaded defense is again quite consistent, and slightly better than the DAE defense. The Dimensionality Reduction defense has commendable performance for the CW and DF attacks, but its accuracy dips quite low when the attack is FGS. When the Fashion-MNIST data is attacked by a CNN, the Cascaded defense and Dimensionality Reduction defense perform quite similarly, but the DAE defense's decrease in performance is very pronounced.

To sum up, when the data is attacked, our defenses increase the accuracy significantly. Among these three, the Cascaded defense is the most consistent in delivering a reliable performance. The DAE defense comes close, but does not always perform well for the FGS attack, and also has poor performance when the Fashion-MNIST data is attacked by a CNN-crafted attack. The Dimensionality Reduction defense performs slightly worse than these two, and also has a tendency to not work well against the FGS attack.

We also show what happens when the defenses are used in the absence of an attack. In general, a stronger defense is expected to compromise the accuracy when there is no attack. To quantify this compromise, we measure the percent decrease in accuracy when using the defense, compared to the accuracy when the defense is not used. When the clean MNIST-Digit data is preprocessed using a defense trained using an FC network, this decrease is quite low, being upper bounded by 0.57%. When the same data is preprocessed using a defense trained using a CNN, the compromise is a little higher, with the percent decrease being not higher than 2.81%. However, the

DAE defense is noticeably less destructive than others, as it decreases the accuracy by only has much as 0.56%. When the uncorrupted Fashion-MNIST data is preprocessed using a defense trained using an FC network, using such a defense reduces the accuracy by as much as 5.8%. However, the Dimensionality Reduction defense performs much better than the rest, decreasing the accuracy by not more than 0.9%. When the same data is preprocessed using a defense trained using a CNN, the DAE defense performs quite poorly, as in the case of attacked data. If considering all defenses other than the DAE defense, the compromise in accuracy is not more than 9.7%. The DAE defense seriously hinders the accuracy without attack, jeopardizing it by as much as 29.23%.

Table 3.2.
Training time of classifiers with and without reduced input dimensions, in seconds

Dataset	**MNIST-Digit**	**Fashion-MNIST**
FC without reduced input dimension	78.82	78.49
FC with reduced input dimension	50.71	50.63
CNN without reduced input dimension	48.53	48.33
CNN with reduced input dimension	13.72	14.01

An added advantage of using this Cascaded defense is that the training time of the classifier is now reduced, since the size of its input layer is smaller. As shown in Table 3.2, the FC network experiences a speedup of 35.66% and 35.49% for the MNIST-Digit and Fashion-MNIST datasets respectively, while the CNN experiences a speedup of 71.73% and 71.01% for the MNIST-Digit and Fashion-MNIST datasets respectively.

Since the tradeoff between accuracy and training time of the classifier with reduced dimension is obvious, it looked like training the classifier less would also result in a more robust classifier. We attempted this by training a classifier till it achieved an accuracy about the same as that of the classifier with reduced input dimension, and found that training the classifier for a lesser number of epochs does indeed lead to it being less sensitive to adversarial perturbation.

4. HIDDEN LAYER DEFENSE

4.1 Motivation

In the Cascaded defense, we were compressing and decompressing the noisy data in the denoising autoencoder, and then compressing it again for dimensionality reduction. We decided to remove this redundancy and to simply use the compressed data generated by the bottleneck layer of the DAE. This has the added advantage of being less computationally expensive while training, compared to the Cascaded defense. This is because the autoencoder for dimensionality reduction does not need to be trained .

4.2 Procedure

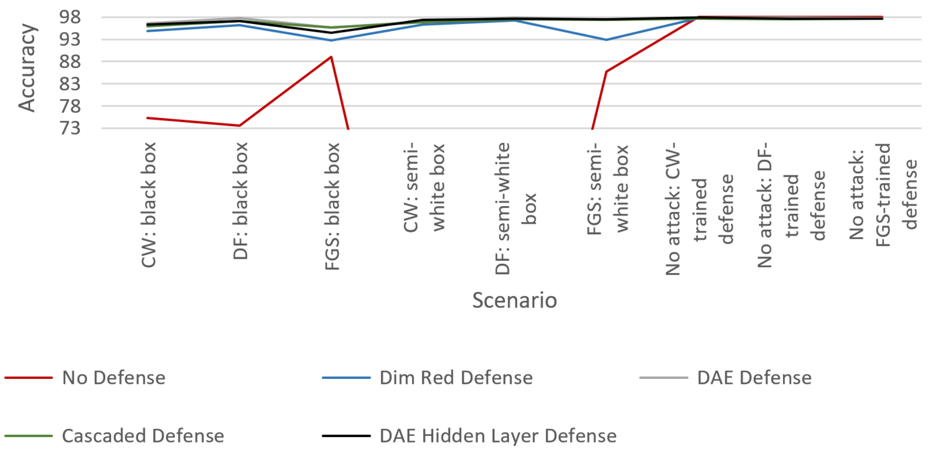
As with the Cascaded defense, we train the DAE to remove adversarial perturbation. However, we do not need another autoencoder to reduce the dimensionality. At test time, we forward propagate the possibly perturbed data through the DAE, but stop at the bottleneck layer. We take the output of the bottleneck layer, and use that as the denoised data with reduced dimension, as shown in Figure 3.1(c). The classifier is trained with this data that is compressed by the DAE. Given that each DAE is trained with an attack generated using a specific attack algorithm using gradients of a specific architecture type, we trained a different classifier corresponding to each DAE. This resulted in six different classifiers.

One may find it hard to see how this data is indeed denoised, since it has not been propagated through the decoder. During the training process, the data is forward propagated throughout the network, and all the weights are updated such that the reconstructed data becomes closer to the clean data. This also means that the weights of the encoder are trained to move the data closer to the clean data. This is why propagating the data only through the encoder indeed gives us a denoised version of the data in a smaller dimension. However, the extent of denoising may be lesser since the encoder was trained to remove perturbation jointly with the decoder.

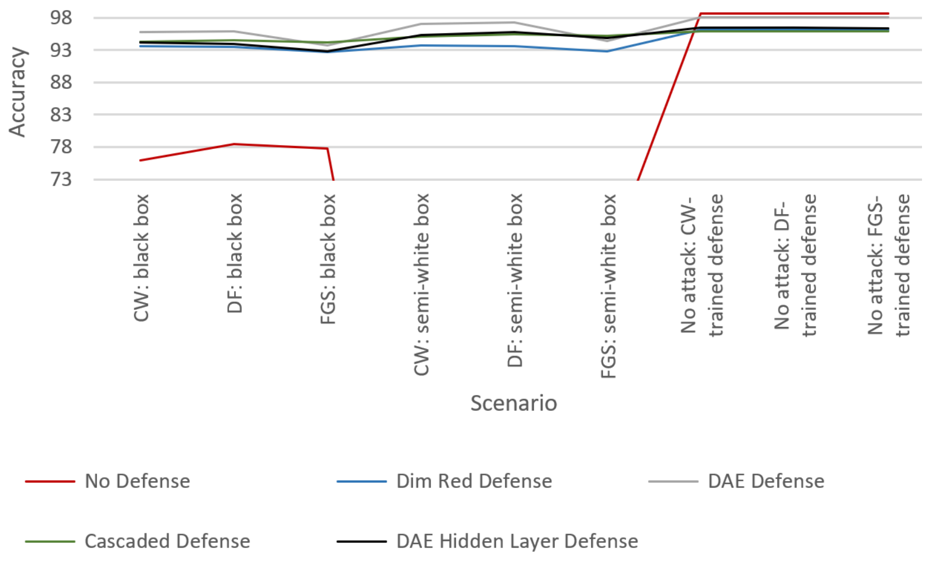
4.2.1 Implementation Details

The architecture and training details of the DAE when implementing the Hidden Layer defense are the same as those of the DAE when implementing the Cascaded defense, as described in Subsection 3.2.1. Regarding the victim classifier, the size of the input layer of the classifier corresponds to the size of the bottleneck layer of the DAE, which is 81. This is the same as the size of the input layer when using the Cascaded defense. All other details about the architecture and training of the victim classifier remain the same as described in Subsection 3.2.1.

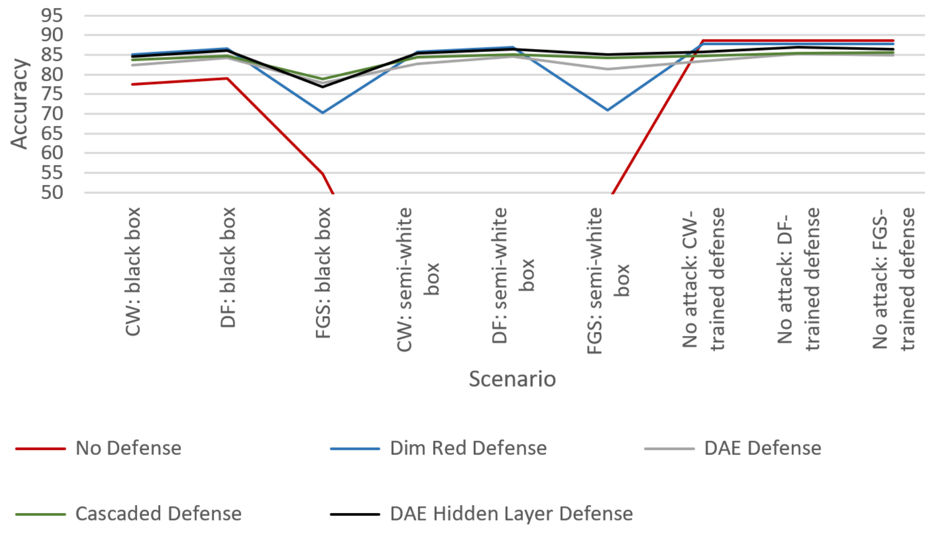
4.3 Results



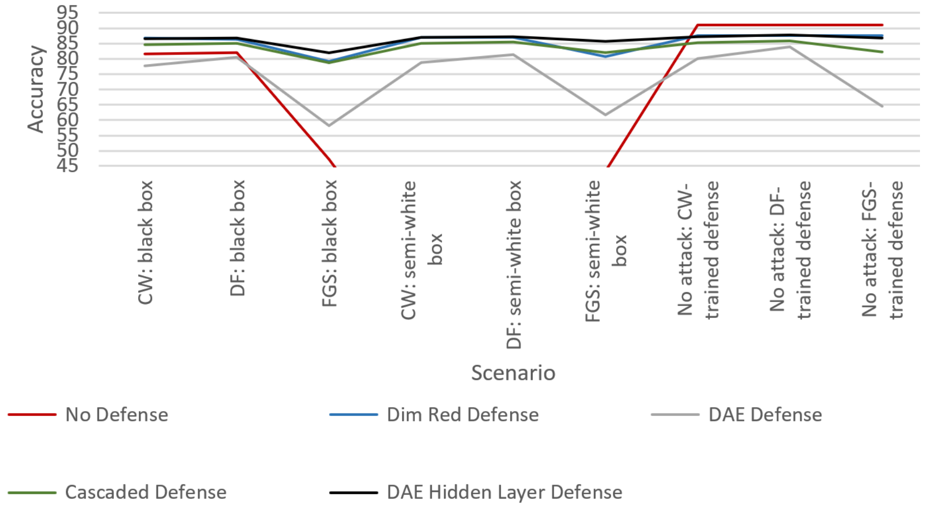
(a) Accuracies when the MNIST-Digit task uses an FC network. Post-attack accuracy in the semi-white box case for (i) CW: 1.24% (ii) DF:1.25%



(b) Accuracies when the MNIST-Digit task uses a CNN. Post-attack accuracy in the semi-white box case for (i) CW: 1.08% (ii) DF:1.09% (iii) FGS: 60.24%



(c) Accuracies when the Fashion-MNIST task uses an FC network. Post-attack accuracy in the semi-white box case for (i) CW: 8.09% (ii) DF: 7.98% (iii) FGS: 47.48%



(d) Accuracies when the Fashion-MNIST task uses a CNN. Post-attack accuracy in the semi-white box case for (i) CW: 8.09% (ii) DF: 7.98% (iii) FGS: 47.48%

Fig. 4.1. Performance of the Hidden Layer defense.

The format of presentation of results for the Hidden Layer defense is similar to the format of presentation of the Cascaded defense, as in Section 3.3. Apart from the performance of the Hidden Layer defense, we also show the performances of the other three defenses in the plots in Figure 4.1 for comparison. Again, in the legends of Figure 4.1, 'Dim Red' corresponds to Dimensionality Reduction.

While defending the MNIST-Digit data, the Hidden Layer defense performs similarly to the Cascaded defense, having an approximately consistent accuracy in different scenarios. The only exception is that in the case of a black box FGS attack, this Hidden Layer defense performs a little worse. While defending the Fashion-MNIST data, the Hidden Layer defense performs slightly better than the Cascaded defense in all cases except when the attack is a black box FGS attack crafted using an FC network.

When the Hidden Layer defense is used in the absence of an attack, its performance is similar to that of the other defenses, sometimes even better. While using the Hidden layer defense, processing uncorrupted MNIST-Digit data using an FC-trained defense and a CNN-trained defense reduces the accuracy by no more than 0.47% and 2.37% respectively. Processing uncorrupted Fashion-MNIST data using an FC-trained defense and a CNN-trained defense reduces the accuracy by no more than 3.1% and 4.6% respectively.

Table 4.1.
Training time of defenses for the Cascaded defense and the Hidden Layer defense, in seconds

Dataset	**MNIST-Digit**	**Fashion-MNIST**
Cascaded defense	333.45	316.38
Hidden Layer defense	268.1	253.48

To sum up, the Hidden Layer defense performs comparably to the Cascaded defense while using less computation. Specifically, the Hidden Layer defense achieves a 19.6% and 19.88% decrease in training time of the defense compared to the Cascaded defense, for the MNIST-Digit and Fashion-MNIST datasets respectively, as shown in Table 4.1.

The recovery of the accuracy using our defenses is especially visible in the semi-white box case, when the accuracy with defense drops to the 1%-10% range for the CW and DF attacks. In fact, we notice that the post-defense accuracy may sometimes be higher in the semi-white box case, compared to the black box case. This makes us wonder if a semi-white box attack is indeed always a smarter choice for an attacker than a black box attack. Perhaps it is important to be prepared to defend against an attack regardless of whether it was constructed using the victim classifier or a different classifier, and regardless of which attack algorithm was used. We make such an attempt in the upcoming chapter.

5. DEFENSE IN AN UNCERTAIN SETTING

5.1 Motivation

We observed that the DAE defense performs satisfactorily when the DAE is trained with an attack generated using the same algorithm and gradients of the same architecture type as the attacker. However, the defender may not always have knowledge about the choices of the attacker. It is not necessary that the attacker will use the most potent attack algorithm known in the literature, which is the CW attack, since it requires quite a lot of computation, and hence a lot of time to be generated. Attempting to generate the CW attack using less time would involve reducing the maximum number of iterations or reducing the number of binary search steps, which would result in an attack that is weak when tested against a different classifier. To make this attack stronger without increasing the maximum number of iterations, the learning rate could be increased. While that would result in a sample that is strongly attacked, the perturbation would be visible. Hence there is no way around the computational cost if one wants to use the CW algorithm generate a strongly attacked sample whose perturbation is imperceptible. Moreover, if this attack is performed during data transmission, a delay in the transmission of data may make the presence of the attacker more obvious. Thus, even if the attacker had the computational resources, they may have reason to choose a different attack. For example, the attacker may find it reasonable to use the FGS attack if they need to compute the attack fast, since the FGS attack involves only one step. The attacker may not find it reasonable to use the DF attack when they are not using the exact trained victim model to generate the attack, since this attack is not very transferable.

Further, it may be difficult for the attacker to gain access to the trained model. Even if the attacker gets this access, they may choose to not use it to generate the attack, for multiple reasons. One reason is that a defense, if it exists, is likely to be trained using an attack generated using the gradients of the trained victim model. If faced with an attack which is also generated using gradients of the same trained network, such a defense will be able to reconstruct the accuracy quite well, as we saw in Section 4.3. Another reason is that sometimes the defender may train multiple victim models to perform the same task, and choose one at random during runtime. So the attacker would not want their attack tailored to only one of those possible victim models, since other victim models could also be chosen. Moreover, the attacker may simply realize that maintaining unpredictability in their choices will make it harder to defend against the attack.

Thus it is important to develop a defense which we can rely on regardless of the choices made by the attacker. We investigate a setting in which the attacker is free to choose any attack algorithm and any architecture whose gradients it will use to generate the attack. We refer to this as a *black box* setting. Note that this threat model is a little different from the typical threat model for a *black box* setting found in the literature. While other threat models portray the attacker as being disadvantaged because of limited knowledge of the victim model, we portray the attacker as using this freedom to be manipulative and unpredictable by not letting the defender make any assumptions about its behavior. It is very natural for the attacker to use uncertainty as a weapon, not only when it has limited knowledge of the setting of victim classifier, but also when it has a significant amount of knowledge. This is because the attacker is aware that if it behaves in the same manner every time, then the defender will eventually find a way around it. At that point, the attacker will simply have to play mysterious.

If the defender knows the details of the attack that the adversary is using, that attack can be defended against, as we showed in Chapters 4 and 3. Similarly, if the attacker knows the details of the defense being used, it can craft an *adaptive attack* to circumvent the defense, for almost any defense, as outlined in [32]. Then the winner is the party that finds out about their opponent's strategy and can act quicker. Preventing the attacker from finding out about the defense strategy is a problem in the area of network security. Discovering details about the attack that the attacker is crafting is also a problem in network security. However, apart from hoping that the attacker does not find out about our defense strategy, and relying on methods to find out how they crafted the attack, the best we can do is attempt to craft a defense that is effective regardless of the choices made by the attacker. We take a step in that direction by training our defense with an ensemble of different attacks which are crafted by varying the attack algorithm and the architecture type of the classifier whose gradient is used to generate the attack.

5.2 Procedure

We examine the behavior of a DAE trained to denoise an attack generated using one attack algorithm or an ensemble of them, and using gradients of one architecture type, or an ensemble of them. For convenience, we henceforth refer to the variables *attack algorithm* and *architecture type* as *defender-determined variables.* As specified earlier, the attack algorithms considered are the Carlini Wagner (CW) algorithm, the Deepfool (DF) algorithm, and the Fast Gradient Sign (FGS) algorithm. The two architecture types considered are Fully Connected (FC) Networks and Convolutional Neural Networks (CNN). The notation *⟨list-of-attack-algorithms⟩-trained defense* is used to refer to a defense trained with attack(s) generated using specific attack algorithms. Similarly, the notation *⟨list-of-architecture-types⟩-trained defense* is used to refer to a defense trained with attack(s) generated using specific architecture types. When we want to specify the architecture type(s) and attack algorithm(s) used to generate attack(s) to train the defense, the notation *⟨list-of-architecture-types⟩-⟨ list-of-attack-algorithms⟩-trained defense* is used. When we want to specify that an attack was generated using a particular attack algorithm and using gradients of a particular architecture type, we refer to it as an *⟨architecture-type⟩-crafted-⟨attack-algorithm⟩* attack.

To test the effectiveness of a defense trained for different choices of the defender-determined variables, we use DAEs as a candidate defense. We train one DAE corresponding to every choice of the defender-controlled variables. The three choices of an attack algorithm give us ${}^3C_1 = 3$ ways to consider only one algorithm, ${}^3C_2 = 3$ ways to consider two algorithms, and ${}^3C_3 = 1$ way to consider all three algorithms. This leaves us with seven ways to make a choice for the attack algorithm(s). The two choices of an architecture type give us ${}^2C_1 = 2$ ways to consider only one architecture type, and ${}^2C_2 = 1$ way to consider both architecture types. Thus, for a dataset where the task can be solved using either FC networks or CNNs, $7 * 3 = 21$ different defenses were trained. For a dataset where the task can be solved using only CNNs, only $7 * 1 = 7$ defenses were trained.

For each defense trained, we tested its effectiveness when a classifier of *architecture type 1* is attacked by an attack generated using gradients of a classifier of *architecture type 2* and using a particular attack type. Here, *architecture type 1* and *architecture type 2* may be the same or may be different. When the task can be solved using either an FC network or a CNN, since the number of possibilities for *architecture type 1* and *architecture type 2* is two each, and there are three possible attacks, the number of scenarios is $2*2*3 = 12$. In addition, we also tested the scenarios in which the victim classifier of a particular architecture type is fed with data that is not attacked. Thus we have a total of fourteen scenarios for a dataset where the classification task can be solved using either architecture type. For a dataset where the classification task can be solved using only one architecture type, we have one scenario corresponding to each of the three attack algorithms, and one scenario corresponding to the case when the data is unperturbed. This results in four scenarios.

5.2.1 Implementation Details

In this chapter, in addition to using the MNIST-Digit dataset and the Fashion-MNIST dataset of grayscale images, we also used the CIFAR-10 dataset [41] of color

Table 5.1.
CIFAR-10 victim CNN architecture.

(Conv 3x3x32, ELU, BatchNorm)x2
Max Pool 2x2, Dropout (rate = 0.2)
(Conv 3x3x64, ELU, BatchNorm)x2
Max Pool 2x2, Dropout (rate = 0.3)
(Conv 3x3x128, ELU, BatchNorm)x2
Max Pool 2x2, Dropout (rate = 0.4)
(Conv 3x3x128, ELU, BatchNorm)x2
Max Pool 2x2, Dropout (rate = 0.4)
Softmax (10 classes)

images. The CIFAR-10 dataset consists of 60,000 32x32 pixel color images of items belonging to one of ten categories such as *airplane*, *cat* and $frog$. It has 50,000 training samples and 10,000 test samples. The classification task is to classify the image into a category.

We use a CNN for the CIFAR-10 classification task, as an FC network would be insufficient. The victim CNN has an architecture as shown in Table 5.1. The accuracy of this classifier is 90.44%. The CIFAR-10 DAE was trained not with perturbations generated using gradients of a single classifier, but with perturbations generated using gradients of multiple classifiers. To obtain variations of this architecture to train the CIFAR-10 defense, an extra (Conv 3x3xz, ELU, BatchNorm) sequence is added after the second, fourth, or sixth such sequence, where z is the number of filters in the convolutional layer preceding the first added convolutional layer. These variations are used only to train the defense, and not to generate attacks to test the defense against. They achieve accuracies of 90.93%, 90.33% and 90.18% respectively. The adversary's architecture is obtained by adding such a sequence after the eighth such sequence. It achieved an accuracy of 89.85%. Note that ELU refers to Exponential

Linear Unit activation and BatchNorm refers to Batch Normalization, both of which are described in the Appendix.

All of these CNNs were trained for 225 epochs with a batch size of 64, with categorical crossentropy loss, and the Root Mean Square Prop (RMSProp) optimizer which starts with a learning rate of 0.001 and decays it with a learning rate of 10^-6. Data augmentation was performed prior to feeding the CIFAR-10 data into the CNN, through rotations of up to 15 degrees, width/height shifts of up to 10% of the original, and horizontal flips.

Table 5.2.
CIFAR-10 DAE architecture.

Conv 3x3x64, ReLU
Conv 3x3x32, ReLU
Max Pool 2x2
Conv 3x3x3, ReLU
Conv 3x3x32, ReLU
Upsampling 2x2
Conv 3x3x64, ReLU
Conv 3x3x64, Sigmoid

The DAE used to denoise CIFAR-10 data has architecture as shown in Table 5.2. As mentioned earlier, it was trained with data that is perturbed using gradients of three different classifiers apart from the victim classifier. It was trained for 150 epochs with a batch size of 256, with MSE loss and Adam optimizer, which uses a learning rate of 0.001.

Details of attack generation remain the same as described in Subsection 3.2.1. One exception is that the learning rate of CW attack was set to 0.3 for the Fashion-MNIST dataset, since a learning rate of 0.1 generated a very weak attack. For the CIFAR-10 dataset, the l_2 norm of the FGS attack was set to 1.7. Examples of such

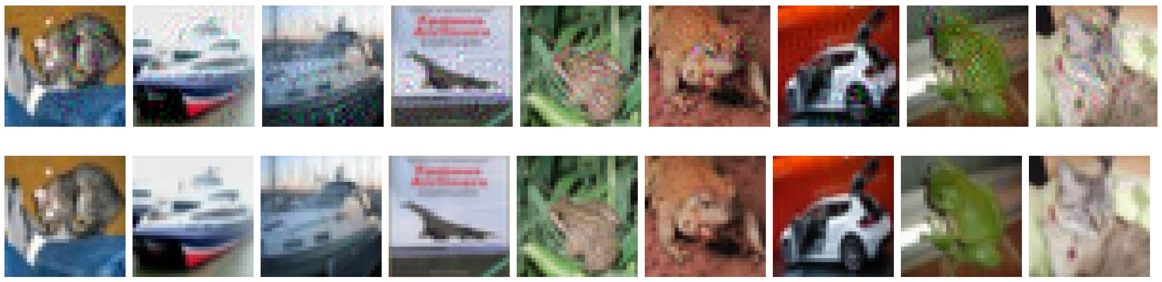


Fig. 5.1. Images from the CIFAR-10 dataset attacked with the FGS algorithm with an l_2 norm of 1.7. The upper row has unperturbed images. The lower row has perturbed images.

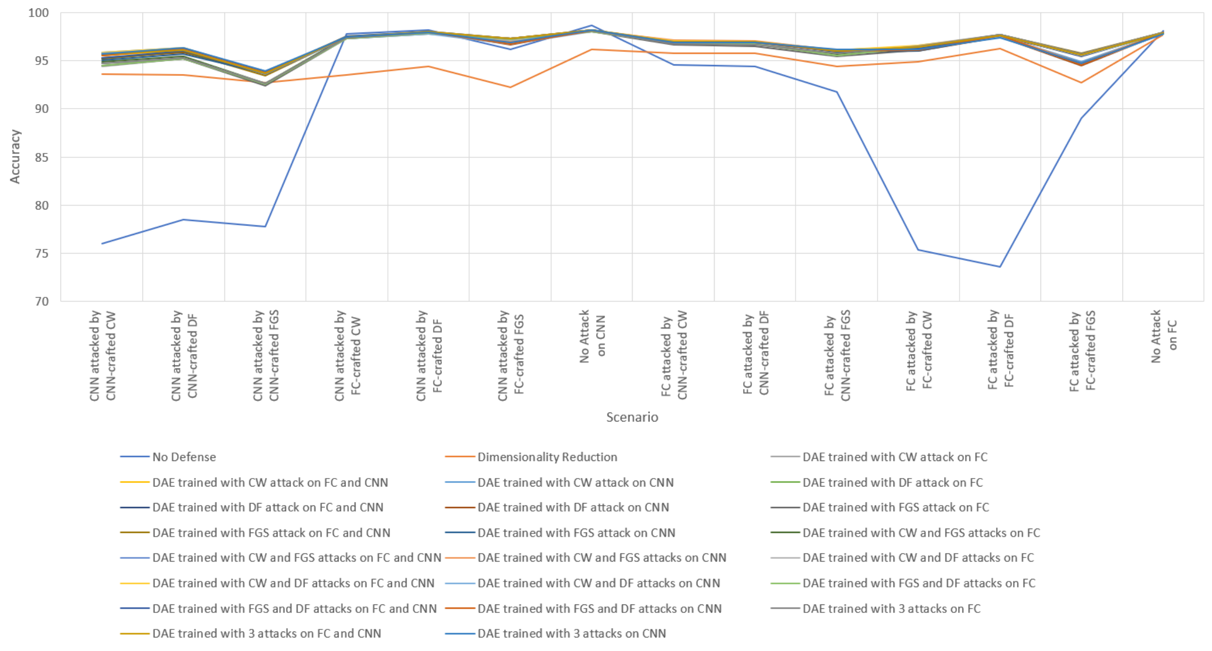
perturbed samples can be found in Figure 5.1. For this dataset, the CW attack was generated with 6 binary search steps, a maximum of 10000 iterations, a learning rate of 0.7, a batch size of 25, an initial constant of 0.001, and the abort_early parameter was set to True.

5.3 Results

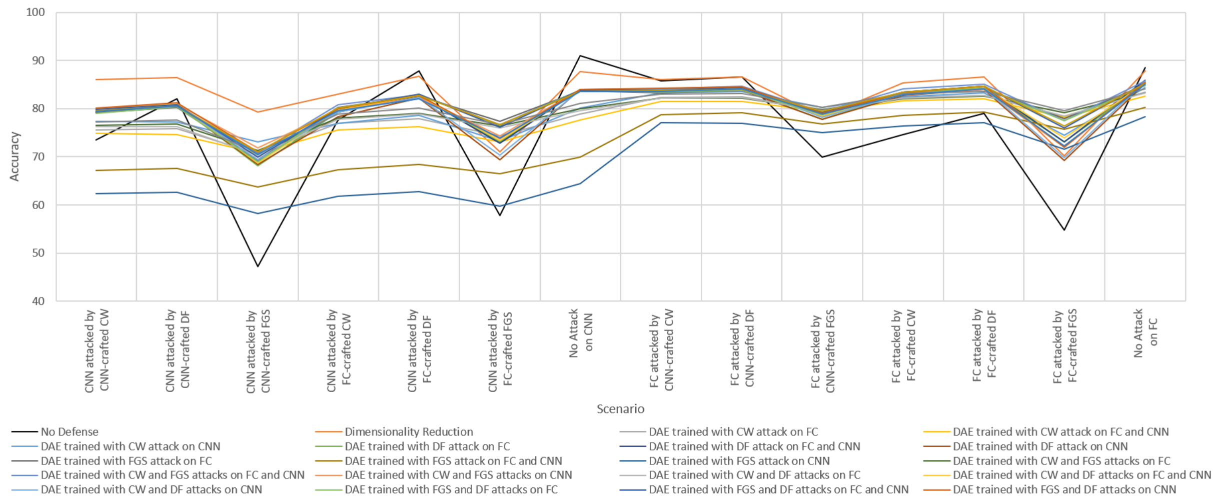
5.3.1 Performance Metric

As shown in Figure 5.2, when the attacker generates an attack using the same architecture type as the victim classifier's, it reduces the accuracy significantly, in which case our defense increases the accuracy quite a bit. This is especially visible for the MNIST-Digit dataset. When the attacker uses a different architecture type to generate the attack, the attack is weaker, and our defense does not have much to do. In some cases when the attack is very weak (example: CNN attacked by FC-crafted DF) or when there is no attack, using our defense may even decrease the accuracy a little bit. This is why it may be a good idea to only use a defense if the test-time accuracy is significantly lower than the validation accuracy. This figure is also important in giving us a sense of what to expect from our defense in any given scenario. No matter how strong the attack is, our defense is always able to recover the accuracy to above 92% for the MNIST-Digit dataset, and to above 73% for the Fashion-MNIST dataset. However, among the differently trained DAEs, some perform better than others in different scenarios. For example, with the Fashion-MNIST dataset, in the scenario of a CNN being attacked by a CNN-crafted DF attack, a DAE trained with the FGS and DF attacks on a CNN increases the accuracy to 81.2%, but a DAE trained with the FGS attack on a CNN is only able to increase the accuracy to 62.6%. Hence, investigating the performance of each defense in a given scenario may go a long way in making a wise choice of a defense.

For a particular defense, we define the *accuracy change* as the change in accuracy from the no-defense case. This is calculated by subtracting the pre-defense accu-



(a) Accuracies for the MNIST-Digit dataset.



(b) Accuracies for the Fashion-MNIST dataset.

Fig. 5.2. Accuracies of all considered defenses in all considered scenarios for the MNIST-Digit and Fashion-MNIST datasets.

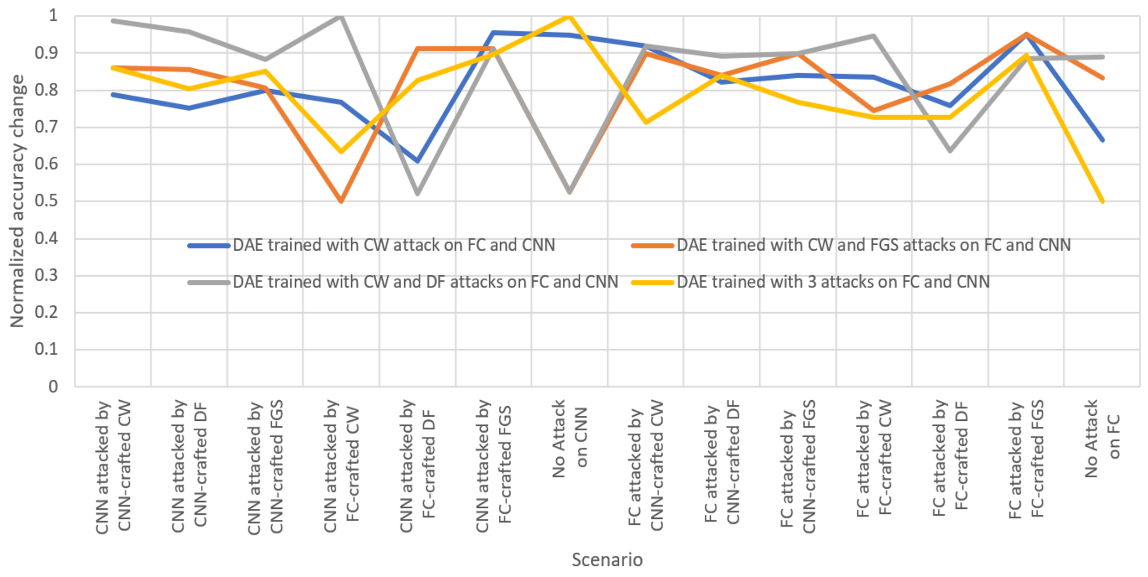
racy from the post-defense accuracy. This accuracy change is positive if the defense increases the accuracy, and negative if it decreases the accuracy. In a particular scenario, to measure the accuracy change resulting from a specific defense relative to the accuracy change resulting from other defenses, we normalize the accuracy changes for all defenses in a particular scenario as follows:

$$normalized\ accuracy\ change = \frac{acc - min(acc)}{max(acc) - min(acc)} \tag{5.1}$$

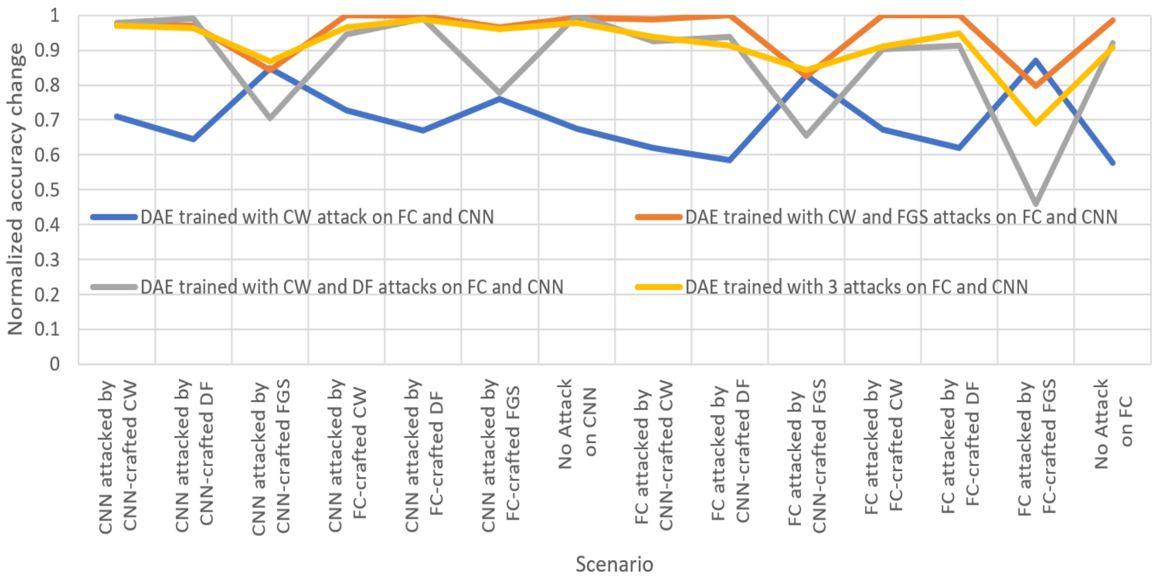
where *acc* is a vector consisting of accuracy changes for each defense corresponding to a particular scenario, whose minimum value is $min(acc)$, and maximum value is $max(acc)$.

This *normalized accuracy change* conveys the extent to which a particular defense is effective in a given scenario, relative to the other defenses. The value of this metric lies between 0 and 1. In a given scenario, it is 0 for the weakest defense, and 1 for the strongest defense. When calculating the normalized accuracy change, we consider only the DAE defenses, and not the Dimensionality Reduction defense.

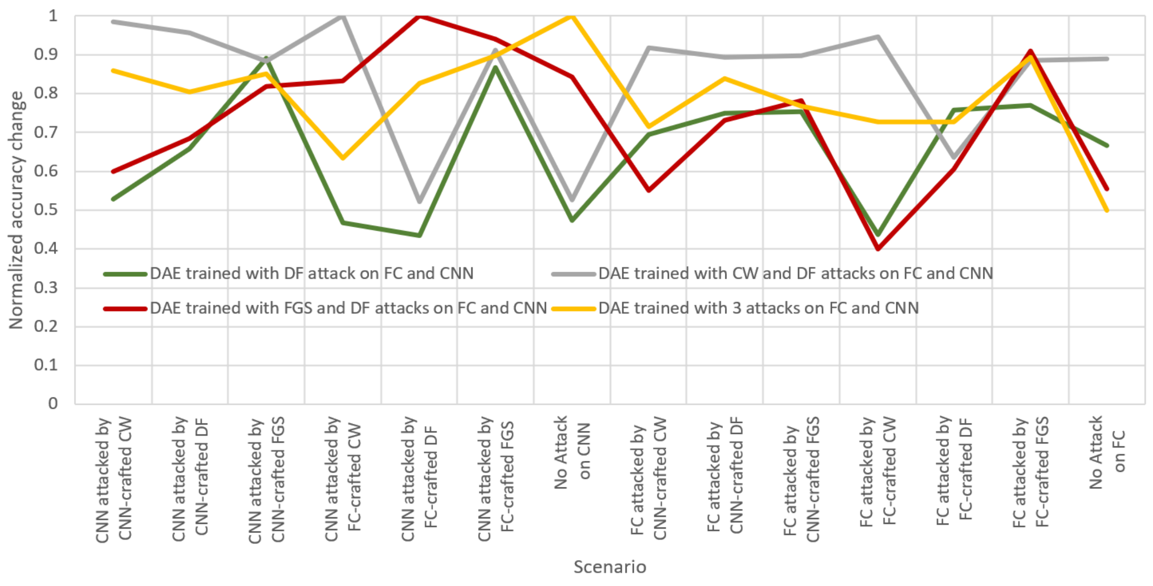
5.3.2 Varying the Architecture Type



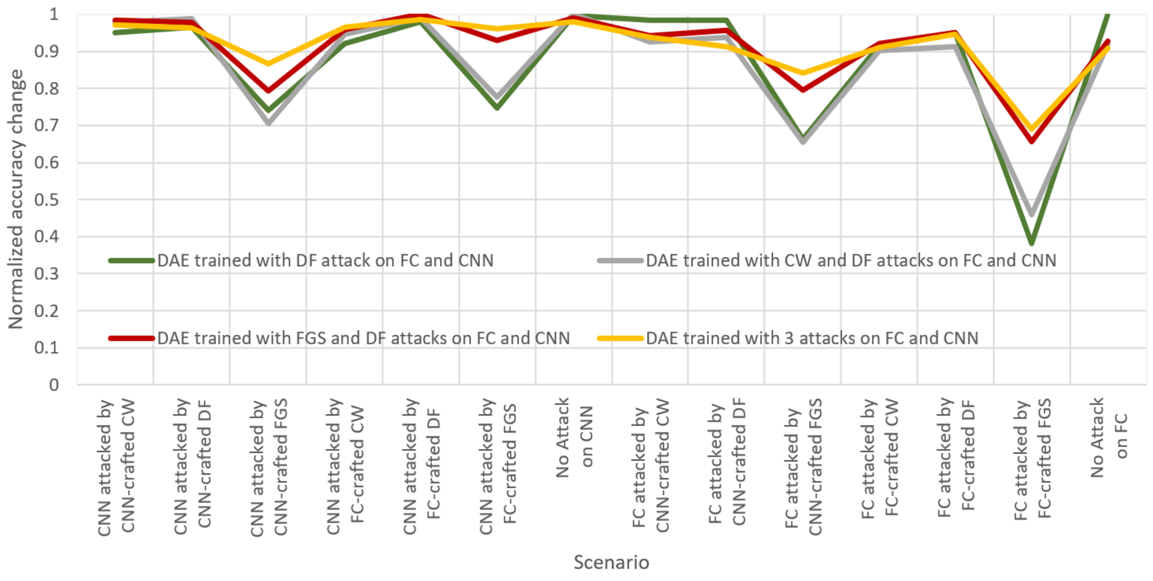
(a) Normalized accuracy changes when using defenses trained with the CW attack for the MNIST-Digit dataset.



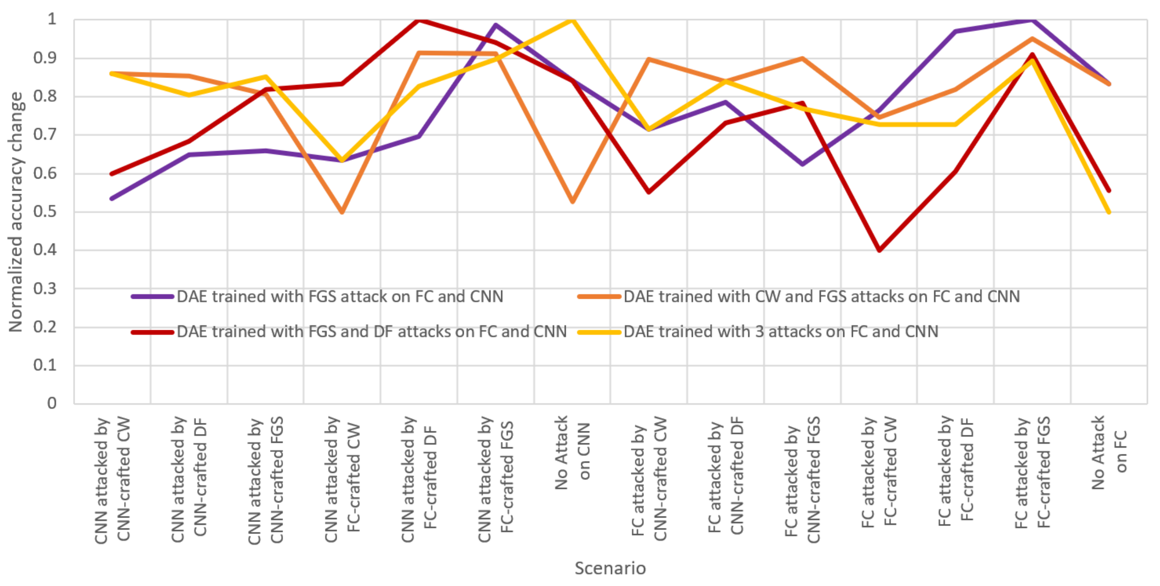
(b) Normalized accuracy changes when using defenses trained with the CW attack for the Fashion-MNIST dataset.



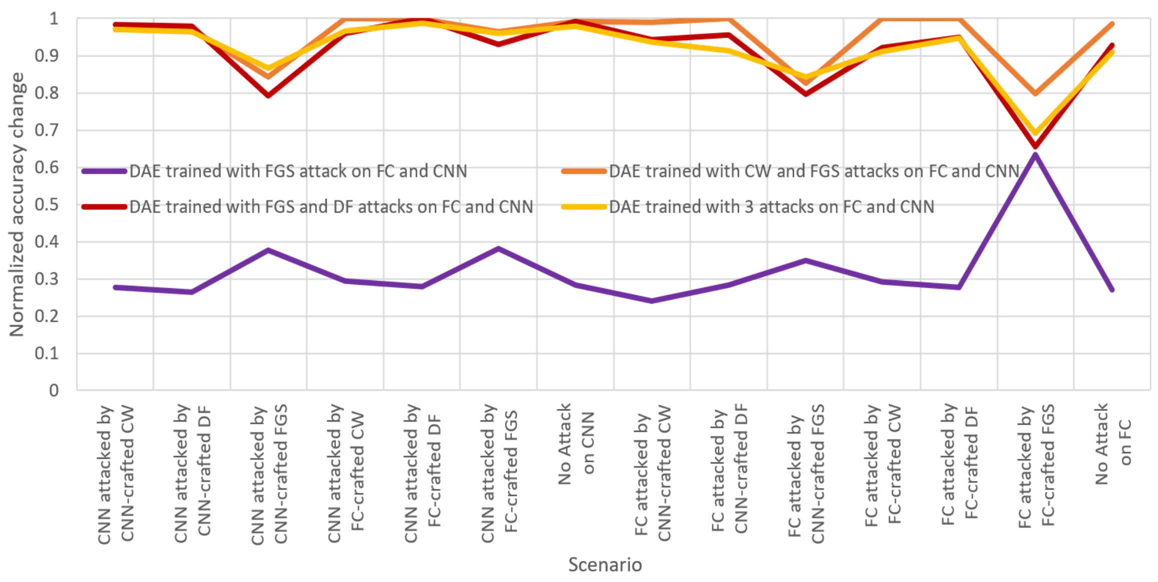
(c) Normalized accuracy changes when using defenses trained with the DF attack for the MNIST-Digit dataset.



(d) Normalized accuracy changes when using defenses trained with the DF attack for the Fashion-MNIST dataset.



(e) Normalized accuracy changes when using defenses trained with the FGS attack for the MNIST-Digit dataset.



(f) Normalized accuracy changes when using defenses trained with the FGS attack for the Fashion-MNIST dataset.

Fig. 5.3. Normalized accuracy changes when using an FC-CNN-trained defense.

Figure 5.3 shows the normalized accuracy changes separately for all defenses in whose training a particular attack algorithm was involved. Since we are considering the MNIST-Digit dataset and the Fashion-MNIST dataset, we consider all fourteen scenarios in which we vary the architecture type chosen by the adversary, the architecture type of the victim network, and the attack algorithm (if any) chosen by the attacker. We only show the performances of the defenses that were trained using attacks generated using gradients of both architecture types. This is because these defenses perform better than defenses trained using attacks generated using gradients of only one architecture type. This is made obvious by the fact that the normalized accuracy changes of the FC-CNN-trained defenses are generally above 0.4. The only exceptions are for the Fashion-MNIST dataset, when a DF-trained DAE is used to defend an FC network against an FC-crafted FGS, and when an FGS-trained DAE is used. This means that the lower normalized accuracy changes correspond to the FC-trained defenses and the CNN-trained defenses.

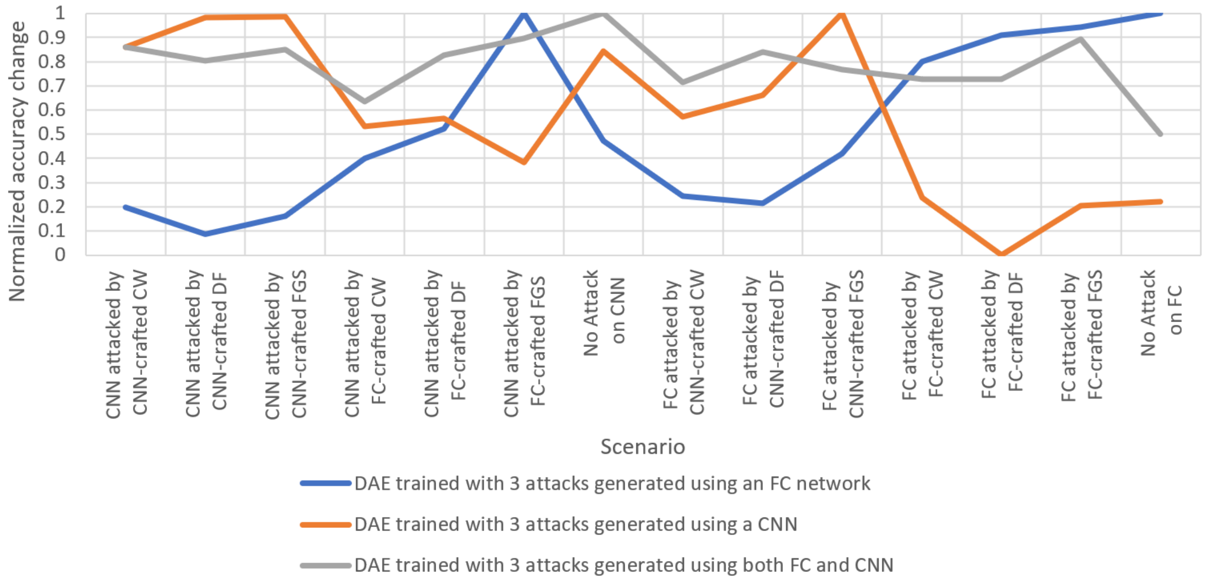
Among all defenses trained with the CW attack, the CW-DF-FGS-trained defense performs the best, as shown in Figures 5.3(a) and 5.3(b). The only scenarios in which this defense does not perform well is when the pre-defense accuracy is high, which may be caused by a weak attack, or by unattacked data. Specifically, this is seen for the MNIST-Digit dataset when a CNN is attacked by an FC-crafted CW attack, and when there is no attack on an FC classifier. When the pre-defense accuracy is so high, this defense would not be even used in the first place. For all other scenarios, the normalized accuracy change of the CW-DF-FGS-trained defense is well above 0.6 for both datasets. The remaining three defenses also deliver reasonable performance, maintaining normalized accuracy changes above 0.5 for both datasets, with one slight exception. For the Fashion-MNIST dataset, the CW-trained DAE's performance is not as good as the other two defenses'.

As shown in Figures 5.3(c) and 5.3(d), among all defenses trained with the DF attack, the CW-DF-FGS-trained defense performs well, for both datasets, as noted earlier. For the MNIST-Digit dataset, the CW-DF-trained defense also has reasonable performance, compared to other defenses. The DF-trained defense's normalized accuracy change drops down to values lower than 0.5 not only when the attack is weak but also when an FC network is attacked by an FC-crafted CW attack. While the FGS-DF defense performs quite well in some scenarios, this defense also has a weak point when an FC network is attacked by an FC-crafted CW attack. For the Fashion-MNIST dataset, both the DF-trained DAE and the CW-DF-trained DAE have a harder time defending against the FGS attack than defending against other attacks. Other than that, all the FC-CNN-trained DAEs in whose training the DF attack was involved perform fairly well.

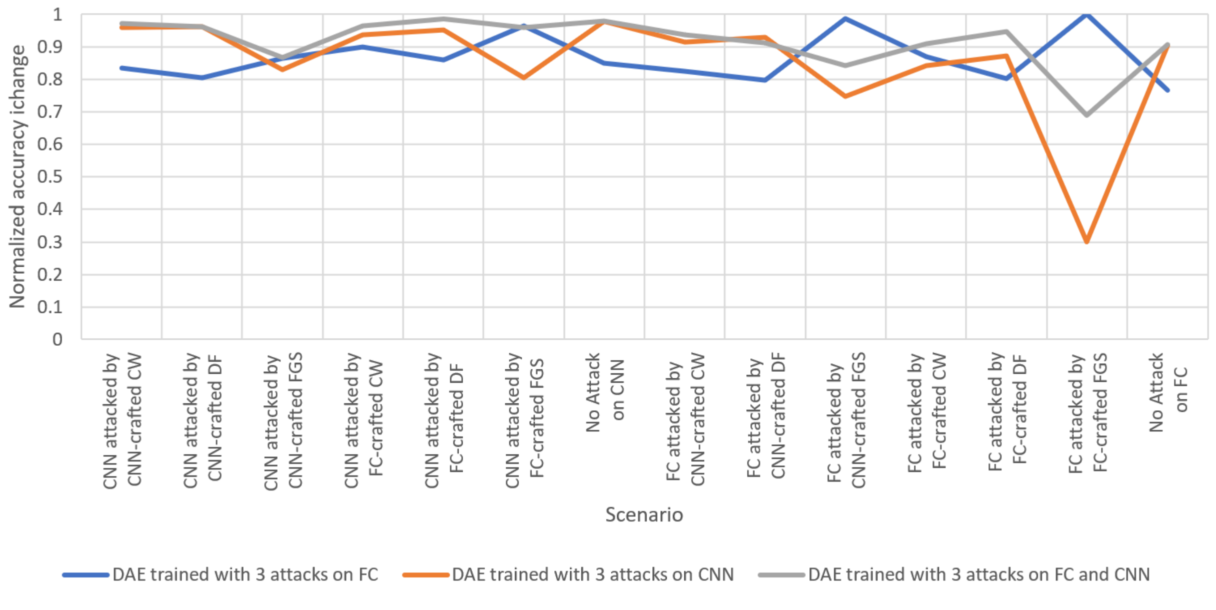
Among all defenses trained with the FGS attack, the CW-DF-FGS-trained defense again performs better among the rest, for both datasets, as shown in Figures 5.3(e) and 5.3(f). For the MNIST-Digit dataset, the remaining three defenses perform reasonably, maintaining normalized accuracy changes above 0.5, except one scenario for the FGS-DF-trained defense, as noted earlier. For the Fashion-MNIST dataset, all defenses have very good and similar performances except the FGS-trained defense, whose performance is less satisfactory, almost always maintaining a normalized accuracy change below 0.5.

In summary, when considering FC-CNN-trained defenses, all defenses trained with the CW attack seem to perform satisfactorily in defending against strong attacks, saving one slight exception, where satisfactory performance is defined as have a normalized accuracy change of more than 0.5. This is especially true of the defenses trained with more than one attack algorithms. Overall, the FC-CNN-CW-DF-FGS-trained defense seems capable of maintaining a reasonable performance consistently, in all scenarios in which a defense is likely to be used.

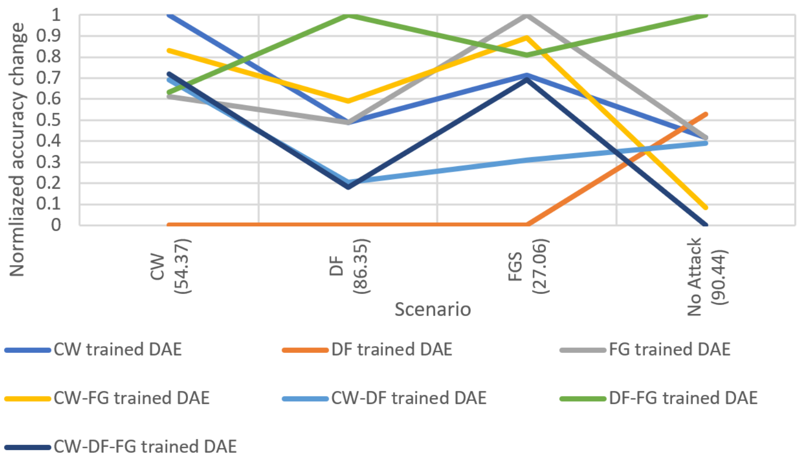
5.3.3 Varying the Attack



(a) Normalized accuracy changes for the MNIST-Digit dataset.



(b) Normalized accuracy changes for the Fashion-MNIST dataset.



(c) Normalized accuracy changes for the CIFAR-10 dataset. The number in parentheses after each scenario description indicates the post-attack accuracy with no defense. This is important to measure how strong the attack is.

Fig. 5.4. Normalized accuracy changes when using a CW-DF-FGS-trained defense.

Figure 5.4 shows the normalized accuracy changes when a defense trained with all three attacks is used. For the MNIST-Digit and Fashion-MNIST datasets, since we have established that a CW-DF-FGS-trained defense performs better than a defense trained using only one or two attack algorithms, we show the performances of only the CW-DF-FGS-trained defenses in Figures 5.4(a) and 5.4(b). We find that the FC-trained defense has a tendency to work well when the attack is generated using gradients of an FC network. Likewise, the CNN-trained defense has a tendency to work well when the attack is generated using gradients of an CNN. These tendencies are especially prominent when using the MNIST-Digit dataset. However, the FC-CNN-trained defense maintains a consistently high normalized accuracy change in all scenarios.

Figure 5.4(c) shows the performances of all defenses trained for the CIFAR-10 dataset. With this dataset, we are only able to consider the four scenarios in which the attacker varies its choice of the attack algorithm (if any). This is because the architecture type cannot be varied for this dataset. The CW-DF-FGS-trained defense performs reasonably in the presence of a strong attack. Many of the other defenses also perform well, or even better. The DF-trained defense is consistently the lowest performing defense in the presence of an attack. Other than that, selectively training a defense with some specific attack algorithm(s) does not seem to influence the effectiveness of the defense. This is not surprising, since we adjusted the hyperparameters of each attack algorithm such that all of them have similar levels of perceptibility, and hence similar levels of strength.

5.4 Conclusion

In conclusion, if there is a possibility of using multiple architecture types to solve the classification task, then the defense should be trained with attacks generated using gradients of different architecture types. Regarding the choice of the attack algorithm used to train the defense, using a ensemble of the three attack algorithms may be likely to perform reasonably when the attack is strong. To determine if the attack (if any) is strong, the test-time accuracy can be compared to the validation accuracy. If the test-time accuracy is significantly lower than the validation accuracy, then using the defense is a good idea.

We make our suggestions with the assumption that all scenarios are equally likely. However, depending on the application, the defender may have some prior belief about which scenarios are more likely than others. In such cases, the results of our experiments in different scenarios may be valuable to determine an appropriate way to train the defense.

6. DISCUSSION

We have proposed a defense that is successful in reducing the training time of the classifier, apart from building robustness against adversarial attacks. Further, we suggested a modification of this defense to reduce the training time of the defense itself, for a small price in the performance of the defense. However, our defense only works well when we train it with attack(s) generated using the gradients of the same architecture type and same attack algorithm that the attack is generated with. In light of the realization that the attacker can vary these parameters unpredictably, we investigated the performance of our defense in multiple scenarios, where the attacker differs their choices. We trained our defense using an ensemble of attacks generated using gradients of different architecture types, and using different attack algorithms. We found that a defense that is trained with attacks generated using gradients of multiple architecture types that can solve the task, and using different attack algorithms, performs reasonably well in all scenarios where the attack is strong.

A caveat of our defense is that it performs questionably when the attack is weak, or when there is no attack. This is why it is important to only deploy our defense when there is a nontrivially strong attack. The most straightforward way to detect such an attack (if any) is to compare the test time accuracy of the classifier to the validation accuracy, without the use of a defense. If the test time accuracy is significantly lower, that indicates the need to use a defense because of the presence of a nontrivially strong attack.

Another contribution of this work is that we identify the scenarios in which Backward Pass Differentiable Approximation (BPDA) may or may not work, as described in Section 2.3. BPDA can easily be overcome if the defender has a variety of defenses available, and randomly chooses one at runtime. As outlined in Section 2.2, there is a large body of existing defenses against adversarial attacks, where each defense has its own strengths and weaknesses. Our DAE defense contributes to this body of defenses, out of which some defenses can be made available for random selection at test time. Such a randomized defense may be viewed by the attacker as a stochastic gradient, and they will attempt to overcome this by applying Expectation over Transformation (EOT) over all the transformations that the defenses perform. However, if we carefully select defenses that perform distinct transformations, the expectation will provide a very inaccurate estimate of the true gradient. Hence the attacker's strategy to overcome such an obfuscated gradient will fail, even if the attacker has access to all of the trained defenses.

Of course, every defense has its own limitations. If the attacker is able to manipulate the seed of the random number generator which determines which defense to use at test time, then they can craft an attack to overcome this defense. However, this would be really hard to do for the attacker. A question that is open to discussion is what a realistic threat model is. This may vary depending on the application, and will only become clearer once adversarial attacks are seen in practice. Considering this, we have kept an open mind about the reasonableness of different threat models, and have experimented with different ways to train and deploy the Denoising Autoencoder defense.

REFERENCES

REFERENCES

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” 2013.

[2] P. Tabacof and E. Valle, “Exploring the space of adversarial images,” in *2016 International Joint Conference on Neural Networks (IJCNN)*, vol. 2016-. IEEE, 2016, pp. 426–433.

[3] S. Jha and E. J. Topol, “Adapting to artificial intelligence: Radiologists and pathologists as information specialists,” *JAMA*, vol. 316, no. 22, pp. 2353–2354, 2016.

[4] M. Abramoff, “Fda clears ai-based device to detect diabetes-related eye problems.(lab notes)(artificial intelligence),” *Clinical Lab Products*, vol. 48, no. 4, p. 6, 2018.

[5] S. G. Finlayson, H. W. Chung, I. S. Kohane, and A. L. Beam, “Adversarial attacks against medical deep learning systems,” 2018.

[6] M. Gopikrishnan and T. Santhanam, “Improved biometric recognition and identification of human iris patterns using neural networks,” *Journal of Algorithms and Computational Technology*, vol. 6, no. 3, pp. 411–420, 2012.

[7] F. Sadikoglu and S. Uzelaltinbulat, “Biometric retina identification based on neural network,” *Procedia Computer Science*, vol. 102, pp. 26–33, 2016.

[8] J. Chen, Z. Mao, W. Yao, and Y. Huang, “Eeg-based biometric identification with convolutional neural network,” *Multimedia Tools and Applications*, pp. 1–21, 2019. [Online]. Available: http://search.proquest.com/docview/2174599557/

[9] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, “A study of the effect of jpg compression on adversarial images,” 2016.

[10] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, L. Chen, M. E. Kounavis, and D. H. Chau, “Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression,” 2017.

[11] W. Xu, D. Evans, and Y. Qi, “Feature squeezing:detecting adversarial examples in deep neural networks,” in *2018 Network and Distributed Systems Security Symposium (NDSS)*, vol. 2018-, 2018.

[12] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille, “Adversarial examples for semantic segmentation and object detection,” *arXiv.org*, 2017. [Online]. Available: http://search.proquest.com/docview/2076083509/

[13] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness may be at odds with accuracy," *arXiv.org*, 2019. [Online]. Available: http://search.proquest.com/docview/2073804650/

[14] X. Li and F. Li, "Adversarial examples detection in deep networks with convolutional filter statistics," in *2017 IEEE International Conference on Computer Vision (ICCV)*, vol. 2017-. IEEE, 2017, pp. 5775–5783.

[15] J. Lu, T. Issaranon, and D. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly," in *2017 IEEE International Conference on Computer Vision (ICCV)*, vol. 2017-. IEEE, 2017, pp. 446–454.

[16] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.

[17] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2016-. IEEE, 2016, pp. 2574–2582.

[18] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," 2016.

[19] 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 (SP)*, 2016, pp. 582–597. [Online]. Available: http://search.proquest.com/docview/1835565498/

[20] T. Na, J. H. Ko, and S. Mukhopadhyay, "Cascade adversarial machine learning regularized with a unified embedding," 2017.

[21] H. Lee, S. Han, and J. Lee, "Generative adversarial trainer: Defense to adversarial perturbations with gan," 2017.

[22] A. N. Bhagoji, D. Cullina, C. Sitawarin, and P. Mittal, "Enhancing robustness of machine learning systems via data transformations," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2018, pp. 1–5.

[23] R. Sahay, R. Mahfuz, and A. E. Gamal, "Combatting adversarial attacks through denoising and dimensionality reduction: A cascaded autoencoder approach," in *2019 53rd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2019, pp. 1–6.

[24] U. Shaham, J. Garritano, Y. Yamada, E. Weinberger, A. Cloninger, X. Cheng, K. Stanton, and Y. Kluger, "Defending against adversarial images using basis functions transformations," 2018.

[25] R. Shin and D. Song, "Jpeg-resistant adversarial images," in *Advances in Neural Information Processing Systems 30*, 2017.

[26] D. Meng and H. Chen, "Magnet: A two-pronged defense against adversarial examples," in *Proceedings of the 2017 ACM SIGSAC Conference on computer and communications security*, ser. CCS '17. ACM, 2017.

[27] F. Liao, M. Liang, Y. Dong, T. Pang, X. Hu, and J. Zhu, "Defense against adversarial attacks using high-level representation guided denoiser," 2017.

[28] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," vol. 27, 06 2010, pp. 807–814.

[29] N. Carlini and D. Wagner, "Magnet and "efficient defenses against adversarial attacks" are not robust to adversarial examples," 2017.

[30] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," *arXiv.org*, 2018. [Online]. Available: http://search.proquest.com/docview/2073468217/

[31] G. Jin, S. Shen, D. Zhang, F. Dai, and Y. Zhang, "Ape-gan: Adversarial perturbation elimination with gan," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2019-. IEEE, 2019, pp. 3842–3846.

[32] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," 2018.

[33] J. Buckman, A. Roy, C. Raffel, and I. Goodfellow, "Thermometer encoding: One hot way to resist adversarial examples," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=S18Su–CW

[34] C. Guo, M. Rana, M. Cisse, and L. van der Maaten, "Countering adversarial images using input transformations," 2017.

[35] G. S. Dhillon, K. Azizzadenesheli, J. D. Bernstein, J. Kossaifi, A. Khanna, Z. C. Lipton, and A. Anandkumar, "Stochastic activation pruning for robust adversarial defense," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1uR4GZRZ

[36] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille, "Mitigating adversarial effects through randomization," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=Sk9yuql0Z

[37] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, "Pixeldefend: Leveraging generative models to understand and defend against adversarial examples," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJUYGxbCW

[38] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[39] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[40] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, and R. Long, "Technical report on the cleverhans v2.1.0 adversarial examples library," *arXiv preprint arXiv:1610.00768*, 2018.

[41] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/ kriz/cifar.html

APPENDICES

A. COMMON ACTIVATION FUNCTIONS USED

Rectified Linear Unit (ReLU): The ReLU function is defined as:

$$ReLU(s) = max(x, s) \tag{A.1}$$

Softmax: For a vector v with m elements, the softmax function for any element v_i is defined as:

$$softmax(v_i) = \frac{exp(v_i)}{\sum_{j=1}^{m} exp(v_j)}, i \in 1, ..., m \tag{A.2}$$

Sigmoid: The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{A.3}$$

Exponential Linear Unit (ELU): Exponential Linear Unit (ELU) activation is defined as:

$$ELU(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & otherwise \end{cases} \tag{A.4}$$

B. COMMON LOSS FUNCTION USED

Categorical Crossentropy: For a dataset with N samples x_i, where $i \epsilon [1, n]$, when the model predicts the probability of sample x_i belonging to class k as $p(y_i, k)$, categorical crossentropy loss is defined as:

$$loss = -\frac{1}{N} \sum_{i=1}^{N} log(p(y_i, c_i)) \tag{B.1}$$

where c_i is the true class label for data sample x_i.

Mean Squared Error (MSE): Given the target label vector $\hat{y}$ and the computed label vector y of length $len(y)$, the MSE loss of y with respect to $\hat{y}$ is calculated as:

$$loss = \frac{||y - \hat{y}||_2}{len(y)} \tag{B.2}$$

C. COMMON OPTIMIZERS USED

Adam: The Adam optimizer works by maintaining an exponentially weighted average v of past gradients, and another exponentially weighted average s of squares of past gradients. Hyperparameters β_1 and β_2 are respectively used to decide how much to change v and s in each step. Mathematically, this can be expressed as:

$$v = \beta_1 v + (1 - \beta_1)(\frac{\partial J}{\partial W}) \tag{C.1}$$

$$s = \beta_2 s + (1 - \beta_2)(\frac{\partial J}{\partial W})^2 \tag{C.2}$$

Here, J is the loss function and W is the weight matrix. Next, to reverse the bias towards zero, we compute $\hat{v}$ and $\hat{s}$ as follows:

$$\hat{v} = \frac{v}{1 - (\beta_1)^t} \tag{C.3}$$

$$\hat{s} = \frac{s}{1 - (\beta_2)^t} \tag{C.4}$$

where t is an index which increments each time v or s is updated. Finally, the weight matrix W is updated as:

$$W = W - \alpha \frac{\hat{v}}{\sqrt{\hat{s}} + \epsilon} \tag{C.5}$$

where ϵ is a small value to avoid division by zero.

Root Mean Square Prop (RMSProp): The RMSProp optimizer works by dividing the learning rate by an exponentially weighted average s of the squares of past gradients. Updating this average requires choosing a hyperparameter β to decide how much to modify the average with the square of the gradient. Mathematically, the weight matrix W is updated as:

$$W = W - \frac{\alpha}{\sqrt{s} + \epsilon}(\frac{\partial J}{\partial W}) \tag{C.6}$$

where J is the loss function, and ϵ is a small value to avoid division by zero. The exponentially weighted average s is updated as follows:

$$s = \beta s + (1 - \beta)(\frac{\partial J}{\partial W})^2 \tag{C.7}$$

D. COMMON LAYERS USED

Batch Normalization: Batch Normalization involves computing the mean μ and variance σ^2 of a minibatch, and transforming each data sample x_i in the minibatch to $\hat{x_i}$ as follows:

$$\hat{x_i} = \gamma(\frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}) + \beta \tag{D.1}$$

where γ and β are hyperparameters to be tuned.