

© 2019 Rishika Agarwal

ADVERSARIAL ATTACKS AND DEFENSES FOR GENERATIVE MODELS

BY

RISHIKA AGARWAL

THESIS

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Computer Science  
in the Graduate College of the  
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Assistant Professor Sanmi Koyejo  
Assistant Professor Bo Li



## ABSTRACT

Adversarial Machine learning is a field of research lying at the intersection of Machine Learning and Security, which studies vulnerabilities of Machine learning models that make them susceptible to attacks. The attacks are inflicted by carefully designing a perturbed input which appears benign, but fools the models to perform in unexpected ways. To date, most work in adversarial attacks and defenses has been done for classification models. However, generative models are susceptible to attacks as well, and thus warrant attention. We study some attacks for generative models like Autoencoders and Variational Autoencoders. We discuss the relative effectiveness of the attack methods, and explore some simple defense methods against the attacks.

*To my parents, for their unconditional love and support  
and my friends, for being my family away from home...*

## ACKNOWLEDGMENTS

I thank my advisors Professor Sanmi Koyejo, and Professor Bo Li, for their guidance and support. This work would not have been possible without their experienced advice, which helped me ask the relevant questions to base my research on. I thank my Mom, whose unconditional love and belief in me motivated me to keep going despite all hardships. I thank my Dad for encouraging me to chase every dream, no matter how big or daunting.

I thank my colleagues and friends (near and far), with whom I held constructive discussions about my work, and about life in general. Special thanks to the friends who proof-read my thesis and pointed out errors and scope for improvements. Whenever I was stuck with obscure errors which I had no clue how to resolve, **stackoverflow** was the oracle which shone the light. So I thank its extremely competent and helpful community! And finally, I thank all my teachers who contributed to my education, from my childhood to my post-graduation (especially my high school computer teacher Mr.P.Tandon, who first showed me the wonderful world of programming, and my undergrad mentor, Prof.Piyush Rai who introduced ML so beautifully in his lectures, motivating me to learn more). I will forever be grateful for the knowledge you imparted to me, and the curiosity it sparked in me.

## TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION . . . . .	1
CHAPTER 2	GENERATIVE MODELS . . . . .	3
2.1	Discriminative vs Generative Models . . . . .	3
2.2	Autoencoder . . . . .	3
2.3	Variational Autoencoder . . . . .	4
2.4	Generative Adversarial Network (GAN) . . . . .	5
2.5	VAE-GAN . . . . .	6
CHAPTER 3	ATTACKS AND DEFENSES FOR CLASSIFICATION MODELS . .	8
3.1	Type of Attacks on Classifiers . . . . .	8
3.2	Problem Formulation for Attack . . . . .	9
3.3	Review of Popular Attack Methods . . . . .	10
3.4	Problem Formulation for Defense . . . . .	11
3.5	Properties of Good Defense Methods . . . . .	11
3.6	Review of Popular Defense Methods . . . . .	11
CHAPTER 4	ATTACKING GENERATIVE MODELS . . . . .	13
4.1	Overview . . . . .	13
4.2	Motivation . . . . .	13
4.3	Attack Model . . . . .	14
4.4	Datasets . . . . .	14
4.5	Evaluation Metrics . . . . .	15
4.6	Understanding the Figures in Results . . . . .	15
4.7	L-BFGS Attack . . . . .	15
4.8	FGSM Attack . . . . .	17
4.9	Carlini-Wagner Iterative $L_2$ Attack . . . . .	24
CHAPTER 5	DEFENSES FOR GENERATIVE MODELS . . . . .	31
5.1	Overview . . . . .	31
5.2	Adversarial Training . . . . .	31
5.3	Binarization (Binary Thresholding) . . . . .	32
5.4	Mean Filtering . . . . .	33
5.5	Discussion . . . . .	48
5.6	Future Work . . . . .	48
REFERENCES	. . . . .	50

## CHAPTER 1: INTRODUCTION

Machine Learning is a field spanning fields of Computer Science and Statistics, which studies algorithms enabling machines to learn different tasks by using extensive data. The motivation behind the field, is to utilize the vast amount of data available to us to make the machines perform these tasks at par, or sometimes even better than humans. A side-product of this is that it also allows us to gain insight into the learning process used by human brains. Today, it is one of the most relevant and powerful technologies, being deployed to every task imaginable.

The earliest work in Machine Learning started around 1950s, when Turing proposed a ‘learning model’ that could learn and become artificially intelligent. Rosenblatt invented the Perceptron in 1957, which formed the basis of the neural networks we use today. The famous Backpropagation algorithm was introduced in 1980s, and paved the way for extensive research in training powerful neural networks. Early 2000s saw the development of traditional ML models like SVM and Nearest Neighbors, while Deep Learning became feasible around 2010, and is used almost everywhere now, for tasks ranging from distinguishing cats from dogs, to beating the world’s best Chess and Go players.

We have progressed rapidly, to the current state where ML is being applied for solving problems in different fields, like Computer Vision, Natural Language Processing, Neuroscience, Biomedical Engineering, etc. It has proved to be a boon to many of these fields, solving complex problems from scratch in a matter of a few hours, and providing insights which were never possible before. However, as with any powerful technology, AI and ML come with their own set of problems. The rampant development of ML algorithms with superior performance and ease of deployment at scale, gave rise to a new field of malicious algorithms, which aim to fool the carefully trained ML systems. This field came to be known as Adversarial ML, as it exploits specific vulnerabilities of the learning algorithms to generate adversarial examples which look harmless/legitimate to a human, but lead the trained ML models to perform in unexpected ways. For example, a classifier trained to classify images as cats or dogs can be fooled by providing it an adversarial image which looks like cat, but has been carefully perturbed such that the classifier predicts it as dog. Such malfunctioning can be detrimental in crucial tasks like image recognition by an autonomous car - an adversarial **STOP** sign can be classified as a **GO** sign, and cause serious mishaps on the road. Due to its severe implications, Adversarial ML became an important field and gained the attention

of researchers in ML and Security. With the ease of attacking the models, there was need to enhance the security of these models by protecting them against well known attacks, as well as from attacks which were not yet known to everyone. This led to studying defense methods for attacks on different ML models.

Almost all adversarial attacks and defenses developed till now are for Classification models. There has been growing interest in studying adversarial attacks for generative models as well. In the next chapter, we look at the difference between discriminative and generative models, and review some common generative models.

## CHAPTER 2: GENERATIVE MODELS

### 2.1 DISCRIMINATIVE VS GENERATIVE MODELS

Discriminative models model the probability  $p(y|x)$ , ie the probability of a label, given the data. We assume a probability distribution, for eg. gaussian/binomial distribution, and estimate the parameters which best fit the training set. Generative models learn the joint probability distribution  $p(x, y)$ , ie, the probability of the data, given a label. Thus, generative models can be used to obtain the label probability by Bayes conditional probability rule  $p(y|x) = p(x, y)/p(x)$ . Since they model the probability of data given label, they can also be used to “generate” or synthesize new data by feeding a label/latent vector, through the probability distribution  $p(x|y) = p(x, y)/p(y)$ . Thus generative models have greater expressive power. However, discriminative models generally outperform generative models on classification tasks. Most classifiers, like Logistic Regression, Neural Networks are discriminative models, since they directly model  $p(y|x)$ . Some examples of generative models are Naive Bayes, Bayes Net, Denoising Autoencoder, VAE, GAN.

### 2.2 AUTOENCODER

Autoencoders (AE) are neural networks trained to reconstruct the input at the output layer. This can be trivially done by learning an identity function if there are no constraints on the number of hidden units, however in AEs, we restrict the number of hidden units in the middle layers to be smaller than the input size, so that the network forms a compressed representation of the input. This is called the latent representation, and the part of the network upto the latent layer is called encoder. The latent vector is then passed through a decoder which is the mirror image (in terms of architecture) of the encoder. The output layer is thus the same size as the input layer, and the output obtained at this layer is compared to the input to compute the loss to train the network. Due to the encoder-decoder architecture of the network, autoencoders are often used to abstract compressed form representation of data, which can be used for other downstream tasks. Autoencoders are not generative models, strictly speaking, because they can't generate a meaningful output without the latent vector obtained from the input. Variational Autoencoders, discussed next are a stochastic version of autoencoder, which are generative.

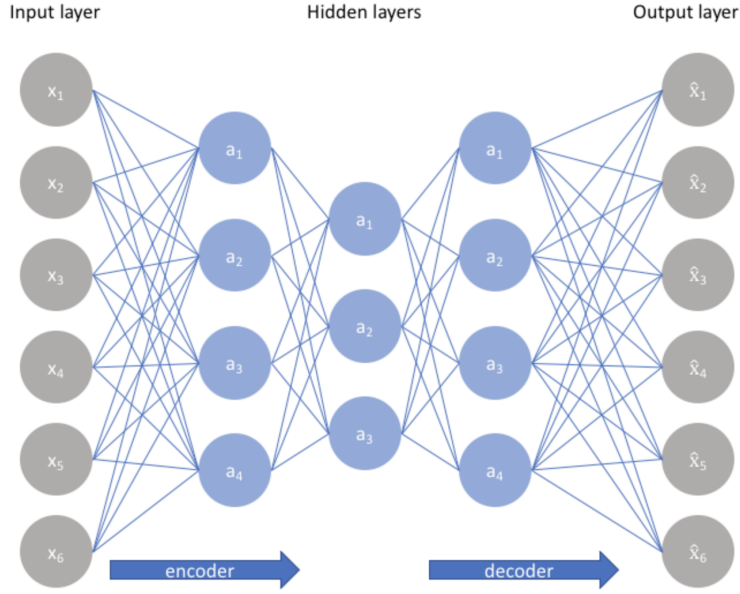


Figure 2.1: Autoencoder architecture  
(source: [www.jeremyjordan.me/autoencoders/](http://www.jeremyjordan.me/autoencoders/))

### 2.3 VARIATIONAL AUTOENCODER

Variational Autoencoders (VAE) are quite similar to autoencoders in their architecture and purpose, however they have better generalizability and generative power which was non-existent in traditional autoencoders. This is obtained by adding a constraint on the encoding network, that forces it to generate latent vectors that roughly follow a unit gaussian distribution. Instead of producing a real-value latent vector which is passed directly to the decoder, VAEs produce a mean vector and a standard deviation vector, which is sampled to obtain the latent vector. This latent vector is then passed through the decoder to obtain the reconstruction of the input. The unit gaussian constraint is required so that we can generate images without any input image - we can randomly sample a vector from a unit gaussian and use it as the latent vector for the decoder. This allows us to generate images out of thin air, unlike autoencoders, which needed to compress an input image to reconstruct an image. There is a trade-off between how closely the latent vector can match a unit gaussian, and how accurate the reconstructions are. This trade-off is left for the network to decide, by training it on a loss function comprising of both the terms - the reconstruction loss, and the KL divergence between the latent layer and unit gaussian distribution. (Eq. 2.1-2.3) While VAEs have proven to be powerful, their reconstructions are somewhat blurry, because of the  $L_2$  loss used in the reconstruction loss. VAE-GANs use GANs to compute the reconstruction



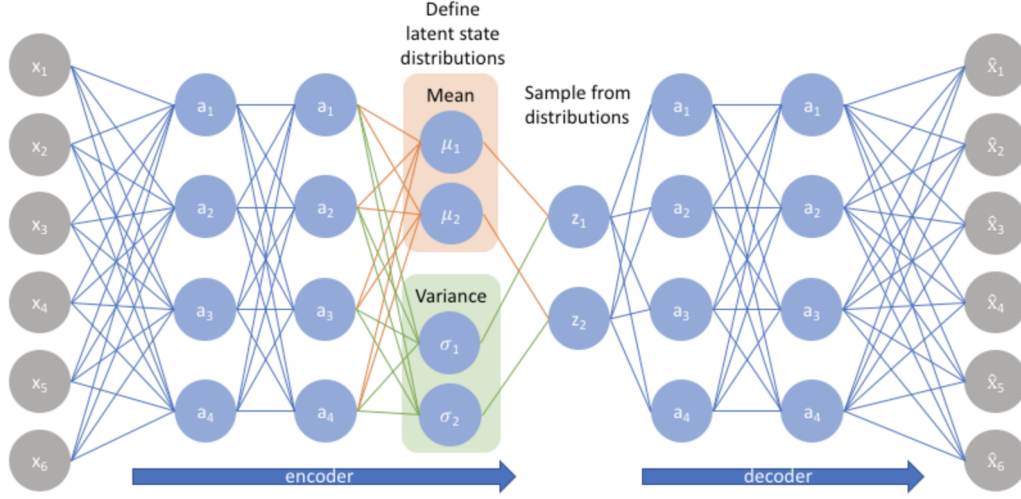


Figure 2.2: VAE architecture  
(source: <https://www.jeremyjordan.me/variational-autoencoders/>)

loss instead, and thus tend to produce better images, as discussed in the next section.

$$l_i(\theta, \phi) = -E_{z \sim q_\theta(z|x_i)}(\log p_\phi(x_i|z)) + \mathcal{D}_{KL}(q(z|x)||p(z)) \quad (2.1)$$

$$= l_{i, \text{like}}^{\text{pixel}} + l_{i, \text{prior}}$$

$$l_{i, \text{like}}^{\text{pixel}} = -E_{z \sim q_\theta(z|x_i)}(\log p_\phi(x_i|z)) \quad (2.2)$$

$$l_{i, \text{prior}} = \mathcal{D}_{KL}(q(z|x)||p(z)) \quad (2.3)$$

$$\mathcal{L} = \sum_{i=1}^N (l_i)$$

## 2.4 GENERATIVE ADVERSARIAL NETWORK (GAN)

Generative Adversarial Networks, popularly referred to as GANs were first introduced in 2014 by Ian Goodfellow and other researchers at the University of Montreal, including Turing award winner Yoshua Bengio [1]. They were received with great enthusiasm in the ML community due to the elegance of the idea of adversarial training, and their power to learn to mimic any distribution of data in any domain : images, music, speech, text. They comprise of two deep networks, involved in a game against each other. One network, called the Generator tries to generate new data samples, while the other network, called discrimi-

nator tries to tell them apart from real world samples, ie, it tries to judge the authenticity of the examples given to it. Thus, the game being played is the generator generating realistic examples which fool the discriminator into believing they are real world examples, and the discriminator trying to maintain a high accuracy of discriminating between generator-synthesized and real-world examples. It can be considered analogous to the game between a counterfeiter and an officer responsible for flagging fake currencies. It is important to note that the game is dynamic, as both the parties are learning on the go with each new example, and they also learn the other party’s methods continuously to improve their own game.

The architecture of a GAN is shown in Fig. 2.3 The discriminator is a simple binary classifier, which takes in an input (generally an image), and predicts the probability of it being fake or real. The generator is similar to the decoder in VAEs, as it takes in a vector, and generates an image from it. In this case, the input vector is a random noise vector. It is generally implemented as an inverse convolutional network. The generator and discriminator try to minimize different loss functions, which are opposing each other (in game-theoretic terminology, they are playing a zero-sum game). The objective function for the discriminator comprises of two terms - the first term is the expectation of correctly predicting a real image as real, and the second term is the expectation of correctly predicting a generator-synthesized image as fake. Thus the discriminator tries to maximize this objective. The generator, on the other hand tries to minimize the loss function in eq.2.4, ie, it wants to minimize the expectation of the discriminator correctly predicting its generated samples as fake. The network is trained by training the discriminator and generator alternatively, through backpropagation. The parameters of one network are fixed while training the other network.

$$\mathcal{L}_{GAN} = V(G, D) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (2.4)$$

$$\min_G \max_D V(G, D) \quad (2.5)$$

## 2.5 VAE-GAN

VAEs produce unrealistic and blurry images due to the way they calculate the reconstruction loss. Element-wise  $L_2$  loss between the pixels of the reconstructed and original images is used. Larsen et. al [2] showed that we can obtain sharper and visually more convincing images by combining VAE and GAN, ie, by replacing the element-wise loss with feature loss

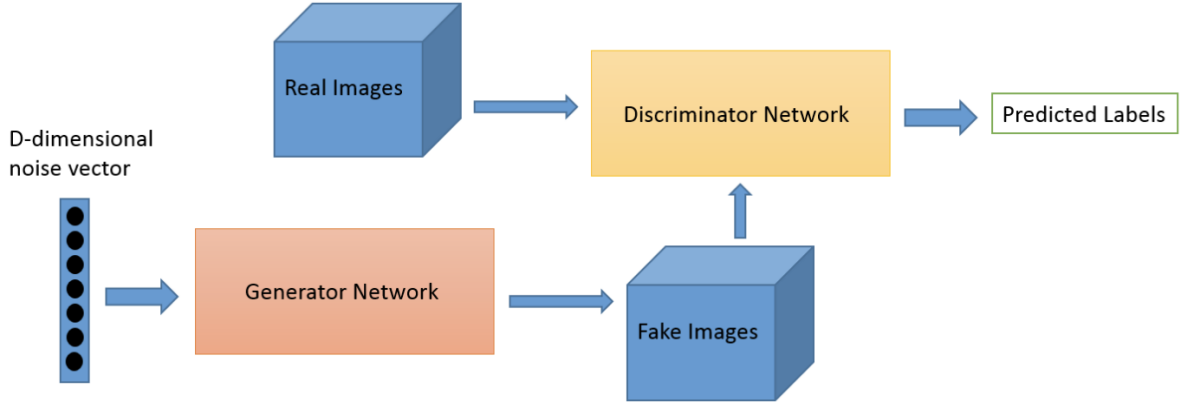


Figure 2.3: GAN architecture  
(source: skymind.ai)

computed by the decoder of a GAN. The model shares the VAE decoder with the GAN generator, ie, they are both the same unit. The discriminator of GAN computes the reconstruction loss between the output of the decoder and the original input. We also add the GAN loss, to enable the network to preserve the style of the inputs. The discriminator reconstruction loss can be seen as the content loss.

$$\mathcal{L} = \mathcal{L}_{prior} + \mathcal{L}_{llike}^{Disl} + \mathcal{L}_{GAN} \quad (2.6)$$

$$\mathcal{L}_{llike}^{Disl} = E_{q(z|x)}(z|x) [\log(p(Disl(x|z)))]$$

and  $\mathcal{L}_{prior}$ ,  $\mathcal{L}_{GAN}$  are given by Eq.1.3 and Eq. 1.5, respectively

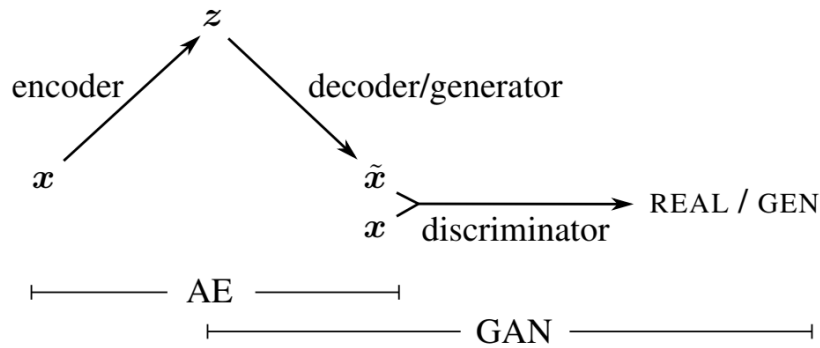


Figure 2.4: VAE-GAN architecture  
(source: Larsen et.al [2])

## CHAPTER 3: ATTACKS AND DEFENSES FOR CLASSIFICATION MODELS

The purpose of an attacker is to fool a carefully trained model (in this case a classifier), either at the train or test time, such that it either learns a poor decision boundary, or misclassifies examples at test time. By fooling a classifier, we mean that the input samples are such that they are practically impossible to differentiate from legitimate examples, but when passed to the classifier, they are mis-classified into an incorrect class. The synthesized adversarial samples need to be imperceptibly different from the legitimate examples, because otherwise a human can easily filter out the adversarial examples from the test set. This means that the noise added to the original sample should have a small enough magnitude to be imperceptible, while also leading to a mis-classification.

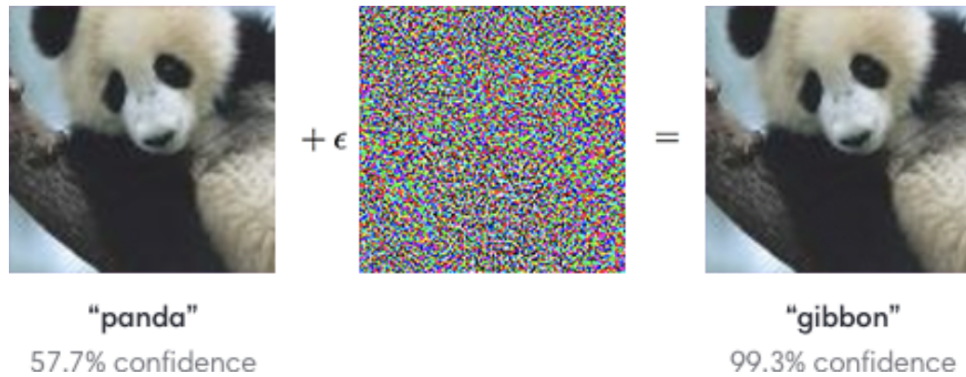


Figure 3.1: An example of adversarial attack which causes a classifier to mis-classify an adversarial image of panda as gibbon (Image courtesy: Goodfellow et.al [3])

### 3.1 TYPE OF ATTACKS ON CLASSIFIERS

Attacks developed so far can be categorized based on different criteria:

1. Based on the sample set they can modify:

- **Evasion Attack** - the adversary perturbs the test samples, to make the trained model misclassify them. The adversary does not perturb the training samples in this case.
- **Poisoning attack** - the adversary perturbs/contaminates some of the training samples, thus affecting the model trained on this set.

2. Based on the purpose of the attack:

- **Untargeted attack** - the adversary is simply interested in contaminating the sample just enough to make the model misclassify it into any class other than the correct class the original sample belonged to.
- **Targeted attack** - the adversary aims to perturb the sample such that the model classifies it to a particular target class, rather than any incorrect class. Targeted attack is thus harder than untargeted attack.

3. Based on the level of access the adversary has:

- **White box attack** - Adversary has all information about the architecture of the classification model and the training procedure and parameters used. It also has access to the training dataset. Most white box attack methods exploit the gradient of the loss function wrt the input, to identify optimal direction of perturbation.
- **Black box attack** - adversary has limited access to the model. Mostly, it does not know the model architecture and training procedure. The training set may/may not be available. On the face of it, black box models might give a sense of security due to restricting the access to the attacker. However, Papernot et.al [4] showed that it is rather easy to attack black box models as well. The general approach to circumvent the restricted information about the model, is to train a local model on either the training set if it is available, or querying the black box model on a set of samples. After training the local model, white box attack methods are used to generate adversarial examples for it, which are then transferred to the black box model.

### 3.2 PROBLEM FORMULATION FOR ATTACK

The problem of generating an untargeted adversarial example for a classifier  $F$ , for the original sample  $X$  can be formulated as follows:

$$X^* = X + \arg \min_{\delta X} (\|\delta X\| : F(X + \delta X) \neq F(X)) \quad (3.1)$$

For targeted attack, with  $y_t$  as the target class:

$$X^* = X + \arg \min_{\delta X} (\|\delta X\| : F(X + \delta X) = y_t) \quad (3.2)$$

Attack algorithms use modifications of this formulation and different optimization methods to generate adversarial examples. Some of the well known white-box attack methods for Deep Neural Networks are L-BFGS, FGSM, Projected Gradient Descent, JSMA, and Deepfool. We will briefly discuss the first two of them, since we will use them in our work on generative models.

### 3.3 REVIEW OF POPULAR ATTACK METHODS

- **L-BFGS:** Szegedy et.al [5] showed in their paper “Intriguing properties of neural networks”, published in 2014, that neural networks are susceptible to adversarial examples, and this was contradictory to the notion that neural networks have high generalization power. They found that these networks learn fairly discontinuous input-output mappings. Thus they could generate adversarial examples by applying hardly perceptible perturbation to the image, such that it maximizes the prediction error. Furthermore, the adversarial examples generated for one network were also successful in fooling another network trained on a different subset of training set (transferability), and with different hyperparameters. They used box-constrained L-BFGS to solve the optimization problem given by Eq.3.1. The adversarial example  $X^*$  is obtained by adding noise  $r$  such that  $X + r$  is still in the input space  $D$ . They were able to generate adversarial examples for linear softmax classifiers with no hidden layers and sigmoidal neural networks with 2 hidden layers and a classifier, trained on MNIST dataset. They were also able to generate adversarial examples for AlexNet.

$$X^* = X + \arg \min_r F(X + r) = l \quad \text{s.t. } (X + r) \in D \quad (3.3)$$

- **FGSM:** Goodfellow et.al [3] debunked the then-held belief that neural networks were susceptible to adversarial attacks due to their nonlinearity and overfitting, and showed that it is indeed the opposite - they are susceptible due to their linearity. They also proposed a fast method to generate adversarial examples which can be used for adversarial training to make the models robust to attacks. The method, called fast gradient sign method linearizes the cost function and calculates its gradient with respect to the input. The adversarial example is then generated according to Eq. 3.4. They were also able to generate adversarial examples by simply rotating the images in the direction of the gradient. An adversarial objective function based on FGSM was used to train

the models, to increase the robustness to adversarial examples.

$$X^* = X + \epsilon \text{sign}(\nabla_x J(X, y_{true})) \quad (3.4)$$

### 3.4 PROBLEM FORMULATION FOR DEFENSE

The job of the defender is to make the model robust to adversarial attacks. Since the defender does not have any control over the inputs faced by the model, the defense is either reactive - developing methods for identifying adversarial examples and rejecting them, or proactive - making the model robust to adversarial examples by giving the correct predictions for the adversarial examples as well:  $\hat{F}(X^*) = \hat{F}(X) = F(X)$ , where  $X^*$  is adversarial example generated by an attack method following the model in Eq. 3.1(targeted) or 3.2(un-targeted),  $\hat{F}$  is the defense model,  $F$  is the original model.

### 3.5 PROPERTIES OF GOOD DEFENSE METHODS

A good defense method should hold certain properties. It should maintain the same performance on the clean dataset, as the original model. It should be computationally efficient, ie, it should not increase the test run time significantly. It should also be memory efficient, ie, it does not require significantly more memory to perform the task, than the original unguarded model.

### 3.6 REVIEW OF POPULAR DEFENSE METHODS

As discussed above, the defense methods can be divided into two categories : reactive or proactive. Among the reactive defenses, adversarial detection and input pre-processing are the common techniques. Among proactive defenses, Adversarial Training and Network Distillation are effective techniques.

- **Adversarial Detection** Instead of attempting to provide the correct prediction for an adversarial example, a defender can attempt to solve an easier problem - distinguishing the adversarial examples from benign ones. SafetyNet [6] uses an auxiliary detection network to classify inputs as adversarial/benign. PixelCNN [7] used the difference in the distribution of adversarial and clean examples for detection.
- **Input Pre-processing** If input adversarial image can somehow be “cleaned” back to the original image, then the model can easily give the correct prediction for the

processed image. Denoising autoencoders and PixelCNN have been effective for this purpose. Cruder pre-processing steps, like binary thresholding or mean filtering can also be used to reduce the effect of the noise added to the adversarial image.

- **Adversarial Training** A new model with the same architecture as the original model, is trained on the original clean samples along with some adversarial samples generated during training time. It can be viewed as a way to regularize the model, by providing it noisy perturbed samples. It has been shown to be effective against single step attack methods like FGSM, but not against optimization based attacks. Further, adversarially trained models are robust against attacks generated for the model itself, but cannot defend against transferred adversarial examples (adversarial examples generated against a normal model).
- **Defensive Distillation** Papernot et.al [4] suggested the use of distillation for training a DNN classifier to make it robust to adversarial examples. Distillation was a technique which was applied earlier for training smaller networks using knowledge from larger networks, to achieve improvement in computational efficiency. However, Papernot et. al proposed a way to use distillation to extract knowledge after training a network and feeding it back to the same network, to make it resilient to adversarial perturbations. First, the DNN is trained in the regular way, on the training set consisting of images and hard class labels. The class probability vectors of this trained network are then used as soft class labels for training the same network again - the network is trained on the original images, but the original hard class labels are replaced by soft class labels (class probability vector for the given image, from the previous training of the network). They showed that this method produced smoother classifiers by reducing their sensitivity to input perturbations. The success rate of adversarial attacks decreased from 95.89% to 0.45% on the MNIST dataset and from 87.89% to 5.11% on the CIFAR10 dataset.



## CHAPTER 4: ATTACKING GENERATIVE MODELS

### 4.1 OVERVIEW

We implement three types of attacks for Autoencoder and VAE models. Some of the attack methods have been discussed in earlier work by Kos et.al [8] and Tobacof et.al [9]. The attacks, evaluated on MNIST and CIFAR-10 datasets are implemented with the `cleverhans` Python library.

### 4.2 MOTIVATION

Kos et.al [8] discuss a scenario in which an attacker might be motivated to attack a generative model. A sender compresses an image using an encoder (for example encoder of an Autoencoder/VAE/VAE-GAN) and sends the compressed latent representation to a receiver who is space and/or time separated. The receiver decodes the latent vector by passing it through a decoder. Consider the case of a bank customer sending an image of a cheque to the bank. A bank representative from the sender's branch encodes the image and sends it to a representative in the head branch, who then decodes it with his decoder. If the customer has malicious intent, they may try to corrupt the image such that the bank representative encoding the image does not suspect the authenticity of the image, yet, when the encoded latent vector for the image is decoded by the second representative, it produces a different image than the input image - for example, if the input cheque image read 100\$, and the decoded image read 900\$. Fig. 4.1 depicts the scenario. There can be some other high risk scenarios which use the same idea of space-separated encoder and decoder to fool carefully designed generative models.

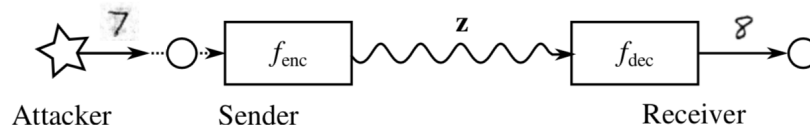


Figure 4.1: Depiction of attack scenario (by Kos et.al [8])

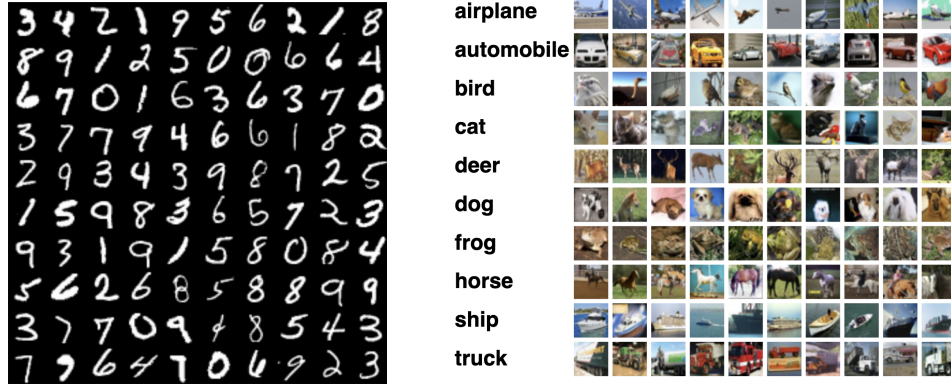


Figure 4.2: Sample images from MNIST and CIFAR-10

### 4.3 ATTACK MODEL

Given an autoencoder model  $M$ , training set  $S_{tr} = \{X_i^{tr}, y_i^{tr}\}$  and test set  $S_{test} = \{X_i^{test}, y_i^{test}\}$ , the aim of an attacker is to pick a source image  $X$  and target image  $X_{tar}$ , and perturb  $X$  by adding some noise to obtain  $X^*$  such that the distance of  $X^*$  to  $X_{tar}$  is minimized, subject to the constraint that  $X^*$  still lies in the valid image space.  $X$  and  $X_{tar}$  are typically selected from the test set.

$$X^* = X + \arg \min_{\delta X} \{dist(M(X + \delta X), X_{tar})\} \quad st \quad X^* \in [0, 1] \quad (4.1)$$

Note that  $M(X) = d(z(X))$  where  $d$  is the decoder and  $z$  is the encoder. The distance between  $X^*$  and  $X_{tar}$  can be represented by different metrics, like  $L_1, L_2, L_\infty$  norms of  $(X^* - X_{tar})$ , or cross entropy distance between  $X^*$  and  $X_{tar}$

### 4.4 DATASETS

We implement attacks on two datasets:

- MNIST - It consists of 50,000 training examples and 10,000 test examples of  $28 \times 28$  black and white handwritten digits.
- CIFAR-10 [10] - It consists of 50,000 training images and 10,000 test images. The images are RGB of size  $32 \times 32$  and belong to one of the following 10 classes : airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

## 4.5 EVALUATION METRICS

An attack is successful if the reconstructed image looks like the target image instead of the original image. Visual similarity is hard to capture numerically, but we use the  $L_2$  distance between the reconstruction and the original image ( $d1$ ) and that between the reconstruction and the target image ( $d2$ ) to capture this. Thus, for a good attack, we should get  $d1 < d2$ . A good attack should also not add too much noise to the original image to obtain the adversarial image, lest it becomes visually recognizable. Another indirect way to measure the success of an attack is through the label prediction for the reconstructions. A classifier is trained on the clean training set, and then used to predict the labels of the reconstructions of the adversarial images. If the prediction is the target label, the attack can be considered to be effective, and if it is the original label, the attack was unsuccessful. Also, if the prediction was neither the target nor the true label, the attack is still a successful untargeted attack.

## 4.6 UNDERSTANDING THE FIGURES IN RESULTS

Referring to the results for all attacks except L-BFGS: for adversarial images, column  $i$  corresponds to the source image belonging to class  $i$ , and row  $j$  corresponds to the target image belonging to class  $j$ . So the image at location  $(i, j)$  is the adversarial image obtained for a source image from class  $i$ , and a target image from class  $j$ . The images lying on the diagonal are thus clean images (no adversarial perturbation), because the source and target images are the same for them. The reconstruction figure shows the reconstructions of the corresponding adversarial inputs. For a perfect attack, each column should display images from 0-9 (for MNIST dataset). The diagonal is again insignificant, as it is the reconstruction of clean unperturbed images.

## 4.7 L-BFGS ATTACK

The L-BFGS attack was built upon the code by Tobacof et.al [9]. They attack the latent layer of the autoencoder, such that the latent representation for adversarial image is close to that of target image, while ensuring that the noise added image is within the minimum and maximum pixel value bounds.

- Loss Function: An autoencoder is first trained on the training set using the usual squared error between the original image and reconstruction.
- L-BFGS attack : L-BFGS attack is used to synthesize adversarial images for the trained

model. The attack consists of a regularization parameter  $C$ , which controls the trade-off between minimization of noise, and closeness of latent representation of adversarial image to that of target image. The best value of  $C$  is found by performing a line search for the maximum value of  $C$  which gives  $\text{adv\_dist} \leq \text{orig\_dist}$ , over a reasonable range of values (where  $\text{orig\_dist} = \text{dist}(\text{original image}, \text{reconstruction of adversarial image})$ ,  $\text{adv\_dist} = \text{dist}(\text{original image}, \text{reconstruction of adversarial image})$ , and  $\text{dist}(a,b)$  is the squared error between  $a$  and  $b$ ). The noise corresponding to this value of  $C$  is added to the original image to get the adversarial image.

$$\begin{aligned} \min_d \quad & \Delta(z_a, z_t) + C\|d\| \\ \text{s.t.} \quad & L \leq x + d \leq U \\ \text{where} \quad & z_a = \text{enocder}(x_t) \\ & z_t = \text{enocder}(x + d) \end{aligned}$$

where the original image is  $x$  and the adversarial image is  $x^* = x + d$ .  $\Delta$  is some distance metric, like  $L_2$  distance, and  $L$  and  $U$  are the lower and upper bounds on the image space pixel values.

- Observations: The attack is very slow, since it searches for the optimal value of  $C$  by line search in every iteration. The adversarial images produced were visibly noisy, however the reconstructions, as shown in Fig 4.3, were quite close to the target images. Adversarial training did not provide any visible improvement in the reconstructions, and surprisingly, was easier to attack than the original model (it had lower attack time, and lower magnitude of noise added).

#### 4.7.1 Implementation Details

This attack was only implemented on MNIST dataset. Due to its very high run time, we did not implement it for CIFAR-10 dataset.

- Autoencoder model :

The encoder consists of 3 Fully Connected (FC) layers, the first two containing 500 neurons each, and the third layer, which gives the latent representation contains 100 neurons. The decoder is exact mirror image of encoder, containing two FC layers with 500 neurons each, and the output layer containing  $\text{img\_ht} \times \text{img\_breadth}$  number of units.

- Parameter Settings :

A batch size of 128 and 10 epochs were used for training the Autoencoder. The line search for the best value of C was done over 100 values in the logspace starting from -20 to 20. For adversarial training, 1000 adversarial images were added to the training set.

#### 4.8 FGSM ATTACK

We used the open source python `cleverhans` library [11] for building FGSM attacks for autoencoders. The library has several attack methods and adversarial training for classification CNNs. We extended it for our study, by including models for autoencoder and VAE, the appropriate loss functions, attack success metrics, and the FGSM attack for the purpose at hand.

- Loss Function: The original autoencoder model was trained to minimize the squared error between the original image and its reconstruction. (For the original model, the original image is the same as the image fed to the encoder, while for the adversarially trained model, for the adversarial examples, the original image is different from the image fed to the encoder)

$$J_{\theta}(x, rec(x)) = \sum_{i,j} (x_{ij} - rec(x)_{ij})^2 \quad (4.2)$$

where  $rec(x) = dec(enc(x))$

- FGSM Attack: The attack uses the negative gradient of the loss wrt the target class probability, to find the perturbation to be added to the original image.

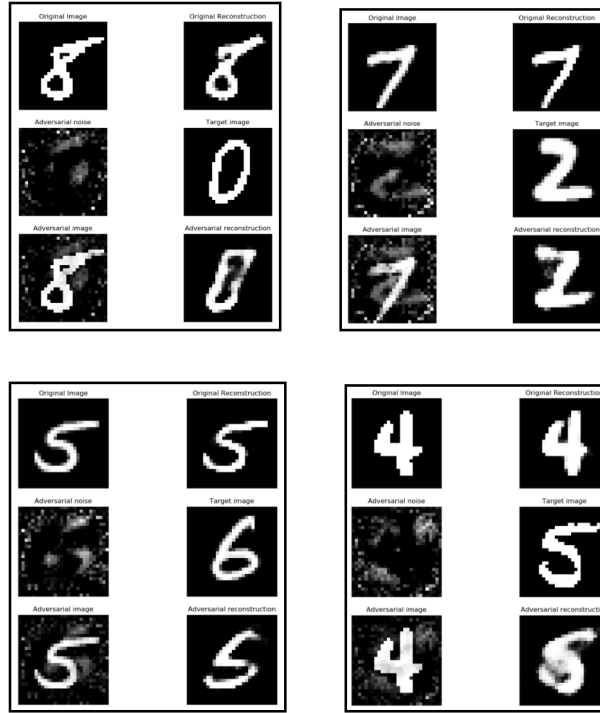
For an untargeted attack, the following equations give the adversarial image  $x'$ :

$$\begin{aligned} \eta &= \epsilon sign(\nabla_x J_{\theta}(x, rec(x))) \\ x' &= x + \eta \end{aligned} \quad (4.3)$$

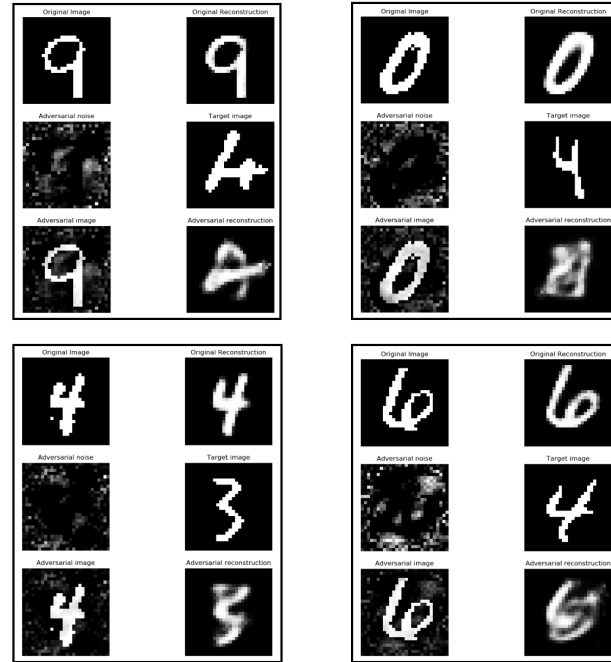
For our study, we studied targeted attacks, thus we modified the attack as:

$$\begin{aligned} \eta &= \epsilon sign(\nabla_x J_{\theta}(x_t, rec(x))) \\ x' &= x - \eta \end{aligned} \quad (4.4)$$

where  $x_t$  is the target image we want the reconstruction of the adversarial image to be close to. The difference is that for targeted attack, we take the gradient of the loss



L-BFGS attack on MNIST Autoencoder



L-BFGS attack on MNIST VAE

Figure 4.3: Examples of L-BFGS attack; Every enclosed square is one example : top left - original image, top right - reconstruction of original image, middle left - noise added to adversarial image, middle right - target image, bottom left - adversarial image, bottom right - reconstruction of adversarial image

function with the target image and reconstruction of original image as inputs, and add negative of that gradient to the original image.

- **Adversarial Training:** Adversarial training uses some clean and some adversarial examples for training the new model. The proportion of clean and adversarial can be controlled by the adversarial coefficient  $\alpha$

$$J_{\theta}(x, x_t) = \alpha J_{\theta}(x, rec(x)) + (1 - \alpha) J_{\theta}(x, rec(x')) \quad (4.5)$$

#### 4.8.1 Implementation Details

##### MNIST

- **Autoencoder model :** The encoder consists of 3 Fully Connected (FC) layers, the first two containing 500 neurons each, and the third layer, which gives the latent representation contains 100 neurons. The decoder is exact mirror image of encoder, containing two FC layers with 500 neurons each, and the output layer containing `img_ht`  $\times$  `img_breadth` number of units.
- **Parameter Settings:** A batch size of 128 and 8 epochs were used for training the Autoencoder. For FGSM attack,  $\epsilon = 0.3$ . For adversarial training, the adversarial coefficient  $\alpha$  had value 0.5.

##### CIFAR-10

- **Autoencoder Model:** It is comprised of 7 convolutional layers, each followed by Batch Normalization, Relu activation (except the last layer, which has sigmoid activation), and MaxPooling/Upsampling for the encoder/decoder. The reconstruction error after 100 epochs is 18.
- **VAE Model:** Convolutional VAE with 3 convolutional layers for the encoder and decoder each. RELU activation is used in each of these layers, while the output layer uses sigmoid activation. The reconstruction error after 100 epochs is 41.
- **Classifier Model:** It is composed of 4 convolutional layers and a dense output layer. Dropouts are used between the layers for regularization. The accuracy on test set after 100 epochs is 72.5%

- Parameter Settings: A batch size of 128 and 8 epochs were used for training the Autoencoder. For FGSM attack,  $\epsilon = 0.3$ . For adversarial training, the adversarial coefficient  $\alpha$  had value 0.5.

## 4.8.2 Results

### MNIST

Fig. 4.4 shows that the reconstructions of the adversarial images are either unrecognizable or close to the original images. The noise added to the inputs is perceivable in some examples, but looks like white noise in most cases. The prediction of the target class was 11% and that of the true class was 47.8%. Fig. 4.5 shows that for VAE, the reconstructions of the adversarial images are perturbed to the target images in just a handful of cases. The noise added to the adversarial images is noticeable, but looks like random white noise. The prediction of the target class was 12.2% and that of the true class was 23.3%. Thus, the attack is more successful against VAEs than Autoencoders. The noise added (3.79) is also less than that added for the autoencoder (5.47)

### CIFAR-10

Fig. 4.6 shows that the adversarial inputs for autoecoder are extremely noisy, and can easily be identified as adversarial. The reconstructions are almost unidentifiable. The classifier prediction of target class is 21.1% and that of true class is 6.8%. The noise added is 15.78. From the figures and the metrics, we can conclude that FGSM is not successful against Autoencoder. Similarly, in Fig. 4.7 the adversarial inputs for the VAE are easily identifiable as adversarial. The reconstructions are predicted as belonging to target class 15.5% and to true class 8.9% of the times. The average distance between the reconstruction and the target images (d2) is much lower than the distance between the reconstruction and the original image (d1).



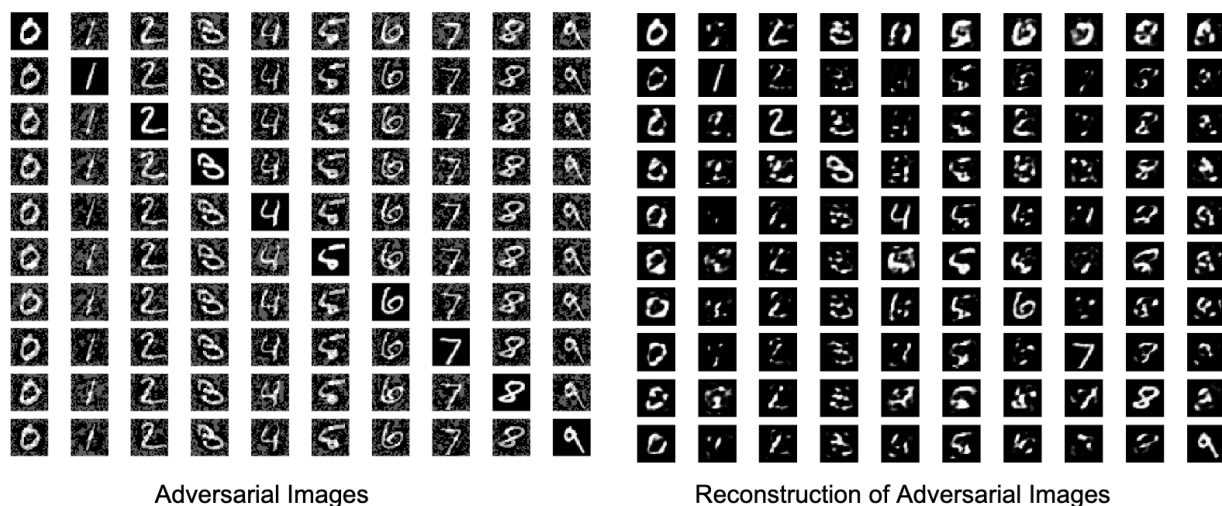


Figure 4.4: FGSM Attack on MNIST Autoencoder

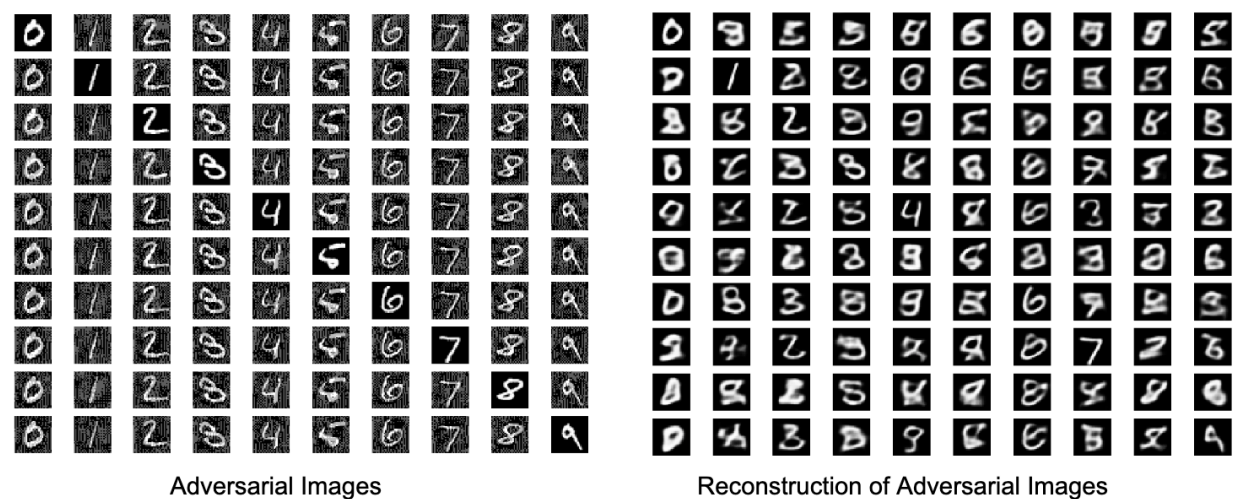


Figure 4.5: FGSM Attack on MNIST VAE

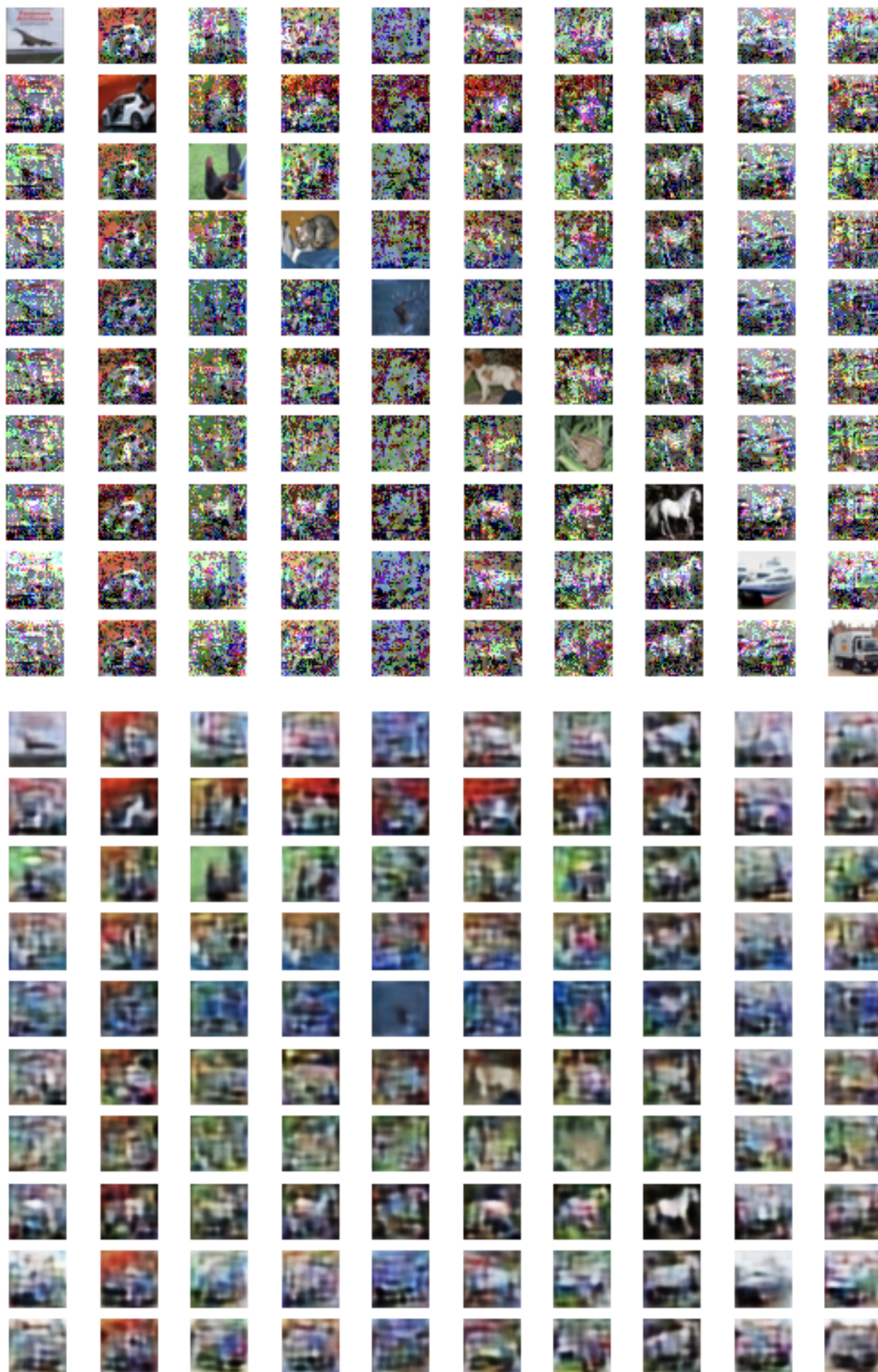


Figure 4.6: FGSM Attack on CIFAR-10, Autoencoder Model: Adversarial Images(top) and their reconstructions(bottom)



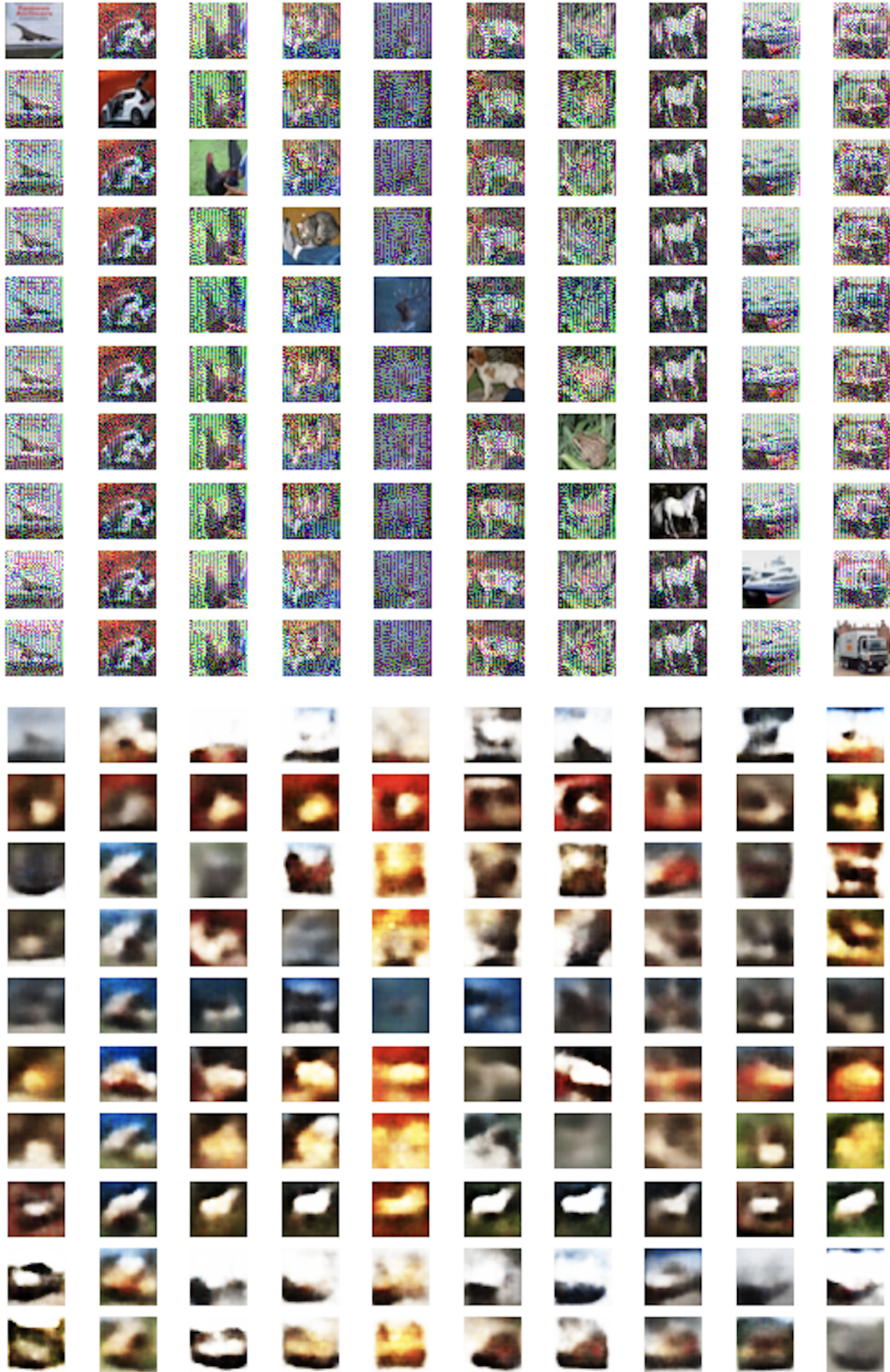


Figure 4.7: FGSM Attack on CIFAR-10, VAE Model: Adversarial Images(top) and their reconstructions(bottom)

## 4.9 CARLINI-WAGNER ITERATIVE $L_2$ ATTACK

This attack is an optimization attack. It minimizes the cost function given by eq. 4.6. The constant  $\lambda$  controls the relative importance given to the reconstruction error and the noise added. Increasing  $\lambda$  should increase the amount of noise added to obtain the adversarial image, in order to obtain a reconstruction closer to the target images, while decreasing it should have the opposite effect. The optimal value of  $\lambda$  is found by performing binary search. We start with an initial value of  $\lambda$ , and update it according to the following rule: if the attack was successful with the current value of  $\lambda$ , we reduce the search space to  $[\text{lower\_bound}, \lambda_{curr}]$ , while if it was unsuccessful, we search in  $[\lambda_{curr}, \text{upper\_bound}]$ . The success of an attack is judged by the prediction of a classifier on the reconstruction of the adversarial image. If the classifier predicts the label as the target label, the attack is said to be successful, and unsuccessful otherwise.

$$\begin{aligned} l_1 &= \|rec(x^*) - x_{targ}\| \\ l_2 &= \|x^* - x\| \\ l &= \lambda l_1 + l_2 \end{aligned} \tag{4.6}$$

A variant of the attack can be obtained by attacking the latent layer instead of the final reconstruction layer. Eq.4.7 shows the cost function for this attack. The optimal value of  $\lambda$  is found in the same manner as discussed above, except that the success of attack is determined by comparing the label of prediction of  $z(x^*)$  by a classifier trained on the latent layer.

$$\begin{aligned} l_1 &= \|z(x^*) - z(x_{targ})\| \\ l_2 &= \|x^* - x\| \\ l &= \lambda l_1 + l_2 \end{aligned} \tag{4.7}$$

### 4.9.1 Implementation Details

#### MNIST

- Autoencoder model : The same as in section 3.5
- Parameter Settings: A batch size of 90 and 8 epochs were used for training the Autoencoder. For Carlini Wagner attack, `num_iterations` varies from 1000 to 10000. `binary_search_steps` varies from 5 to 10.

## CIFAR-10

- Autoencoder model: The same as in section 3.5
- VAE model : The same as in section 3.5
- Parameter settings: A batch size of 90 and 100 epochs was used for training the Autoencoder. The classifier was trained for 80 epochs, with RMSProp optimizer. The number of binary search steps was fixed at 1 (which means no binary search was performed and the initial constant was used throughout)

### 4.9.2 Results

## MNIST

Figure 4.8 shows results for selected values of the parameters `num_iterations` and `binary_search_steps`. The figures suggest that the attack is successful, as the reconstructions resemble the target images in most cases. Some source classes (like 2,7) are easier to perturb than others (like 1,3,9), and some target classes (like 1,5,8,9) are easier to perturb to. The classifier prediction on the reconstructed images belonged to the target class for all source and target classes, giving a 100% target class accuracy. The noise added was 3.94, and the average distance between the reconstructions and original images was almost equal to the average distance between the reconstructions and target images ( $d1 \approx d2$ ). The VAE model (Fig. 4.9) was harder to attack, and the predictions belonged to target and source class 27.7% of the times each.

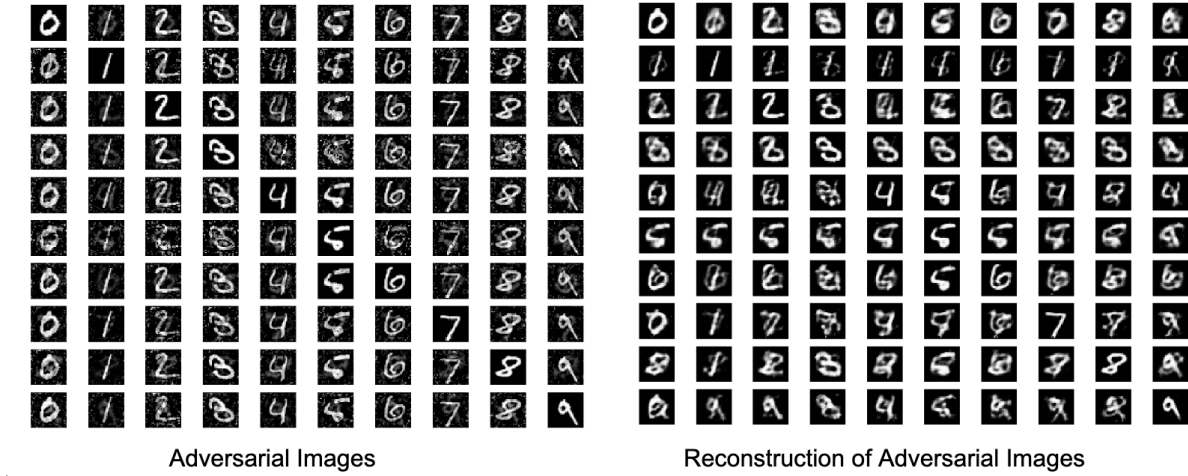
## CIFAR-10

Fig. 4.10 shows the results of CW attack on Autoencoder model. Most of the reconstructions still resemble the original images . The noise added to the images is perceivable for some of the examples, but for most of them it is low. Table 4.2 shows that the classifier prediction for the target class is 31.1% and that for the true class is 25%. The average noise added is 5.29, and the distance of the reconstruction from the target images is significantly higher than the original images. Thus, the attack is not very effective against Autoencoder. For VAE, the reconstructions are blurry, but the attack was successful in causing most of the reconstructions to resemble the target images. The noise added to the adversarial images is imperceptible in most cases. Table 4.2 shows that the classifier prediction for target class

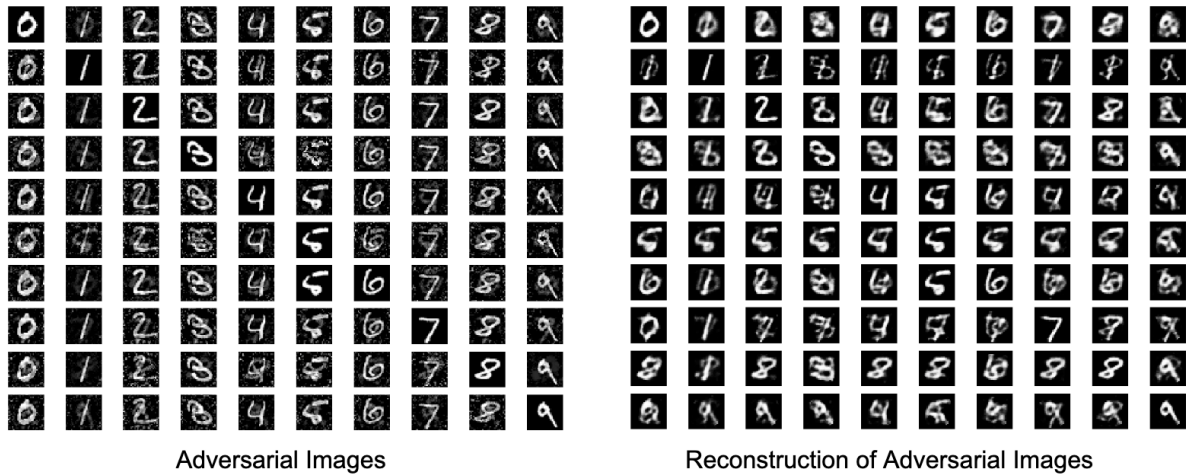
is 70%, compared to 7.8% for the true class. The distance between the reconstruction and target image is lower than that between the reconstruction and original image ( $d_2 < d_1$ ). Thus the attack is successful against VAE.

DATASET	MODEL	ATTACK	METRIC	
MNIST	AE	FGSM	d1	47.69
			d2	59.61
			noise	5.47
			class_acc_clean	0.97
			class_acc_adv_target	0.11
			class_acc_adv_true	0.478
		CW	d1	41.19
			d2	40.37
			noise	3.94
			class_acc_clean	0.97
			class_acc_adv_target	1.0
			class_acc_adv_true	0.0
	VAE	FGSM	d1	61.56
			d2	80.97
			noise	3.79
			class_acc_clean	0.97
			class_acc_adv_target	0.122
			class_acc_adv_true	0.233
		CW	d1	54.73
			d2	46.27
			noise	3.93
			class_acc_clean	0.97
			class_acc_adv_target	0.267
			class_acc_adv_true	0.267

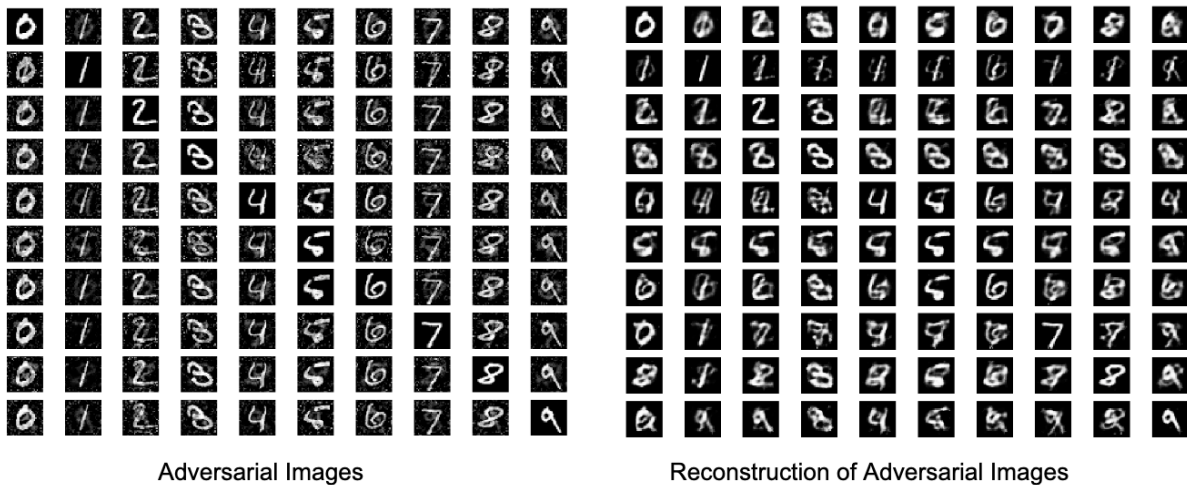
Table 4.1: Results of Adversarial attacks on MNIST dataset



Iter = 1000, Binary Search steps = 5



Iter = 1000, Binary Search steps = 10



Iter = 10000, Binary Search steps = 5

Figure 4.8: CW Attack on MNIST Autoencoder for different values of iterations and binary search steps

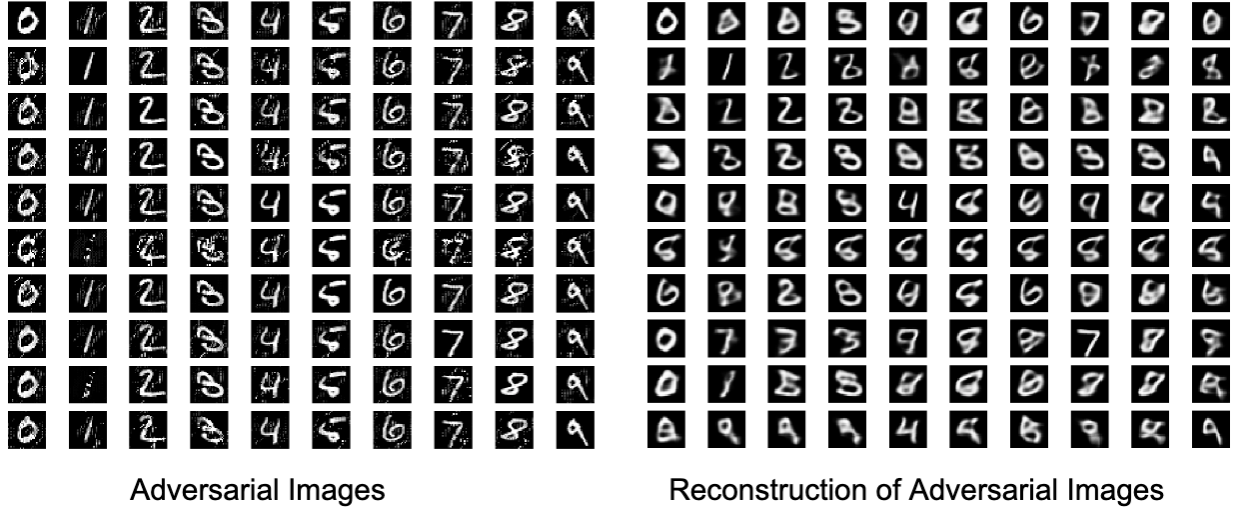


Figure 4.9: CW Attack for MNIST VAE and their reconstructions for Iter = 10000, Binary Search steps = 5

DATASET	MODEL	ATTACK	METRIC	
CIFAR-10	AE	FGSM	d1	149.60
			d2	144.25
			noise	15.78
			class_acc_clean	0.72
			class_acc_adv_target	0.211
			class_acc_adv_true	0.068
		CW	d1	54.97
			d2	261.85
			noise	5.29
			class_acc_clean	0.72
			class_acc_adv_target	0.311
			class_acc_adv_true	0.25
	VAE	FGSM	d1	575.38
			d2	235.33
			noise	15.48
			class_acc_clean	0.72
			class_acc_adv_target	0.155
			class_acc_adv_true	0.089
		CW	d1	250
			d2	147
			noise	7.87
			class_acc_clean	0.72
			class_acc_adv_target	0.7
			class_acc_adv_true	0.078

Table 4.2: Results of Adversarial attacks on CIFAR-10 dataset



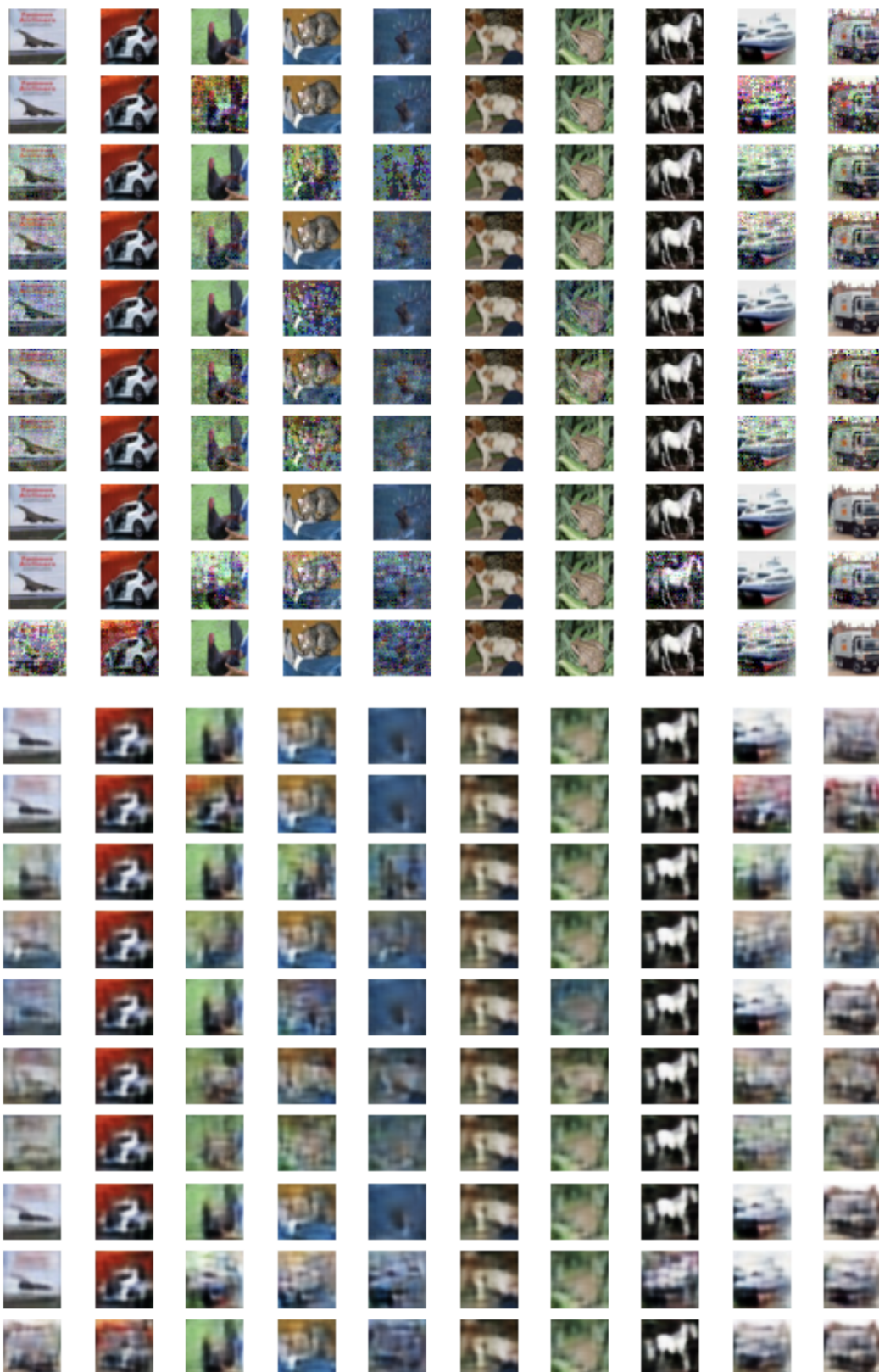


Figure 4.10: CW Attack on CIFAR-10, Autoencoder Model: Adversarial Images (top) and their reconstructions (bottom)

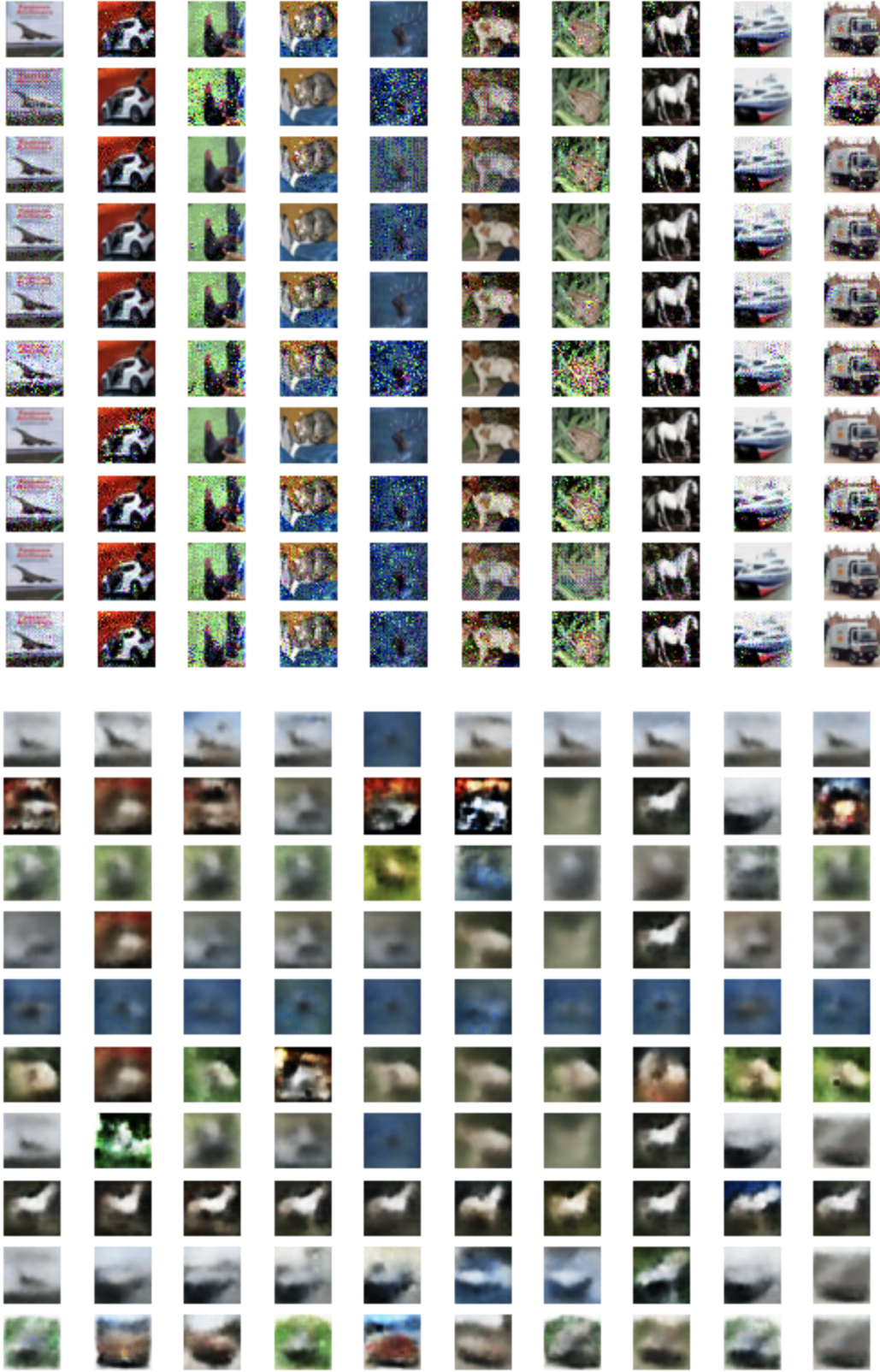


Figure 4.11: CW Attack on CIFAR-10, VAE Model: Adversarial Images(top) and their reconstructions(bottom)

## CHAPTER 5: DEFENSES FOR GENERATIVE MODELS

### 5.1 OVERVIEW

We tried three defense methods for the attacks discussed in the previous chapter - Adversarial Training, Binarization, and Mean Filtering. The latter two are simple image pre-processing techniques, and as we will see, they are also the most effective defenses.

### 5.2 ADVERSARIAL TRAINING

Adversarial training refers to the training of a new model, similar in architecture as the original model, but on a training set comprised of the clean examples from the original training set and adversarial examples generated by the attack on the original model. By teaching the model to predict the desired true label (for a classification model), or reconstruct the true source image (for our case of generative model), we hope that the model will become robust to adversarial examples, and thus perform better against new adversarial examples generated for it. In case of classification models, adversarial training has been shown to be effective against single step attacks like FGSM, but not so effective against iterative attacks. We observe a similar result for our attacks on generative models- it proves to be effective only against the FGSM attack, and ineffective against L-BFGS and iterative CW attacks.

#### 5.2.1 Results

##### MNIST

Against FGSM attack on Autoencoder, it reduces the prediction of target label from 11% to 5.56%, and increases the accuracy of the true label from 47.8% to 78.5%. The new model was trained using the modified cost function given in Equation 4.5, with the adversarial coefficient  $\alpha = 0.5$ .

For CW attack on Autoencoder, adversarial training does not affect the target and true label predictions at all. It does, however, provide marginal improvement in d1 and d2, from 41.19 to 39.9 and 40.37 to 36.13. Thus, it seems that adversarial training provided some regularization, at best.

## CIFAR-10

It increases the prediction of the target label from 31.1% to 34.4% and decreases prediction of true label from 25% to 20% for CW attack on Autoencoder. Thus, ironically, adversarial training makes the model even more susceptible to attacks in this case. On VAE, however, it decreased target label prediction from 70% to 55% and increased true label prediction from 7.8% to 26.7%

### 5.3 BINARIZATION (BINARY THRESHOLDING)

The adversarial images are produced by adding some noise to the original images. Visually, it seems that the noise looks faint enough to be effectively removed by binarization. Thus, we expect that preprocessing the input images by binarization before passing them to the Autoencoder would be a good way to reproduce the original images. Fig. 5.1-5.4 show the adversarial images after binarization, and their reconstructions.

#### 5.3.1 Results

## MNIST

Tables 5.1 and 5.2 show the metrics after Binarization for MNIST. It decreases the target prediction to about 1% and increases the true label prediction to about 90% for all the cases, except CW attack on VAE model, where it raises the prediction to 63%. Visually also, it is a strong defense against both attacks for both the models.

## CIFAR-10

Tables 5.3 and 5.4 show that binarization increases the true class prediction to about 14% for attacks against autoencoder and 4.4%, 18.9% for FGSM and CW attacks respectively against VAEs. It is not a good defense for CIFAR-10, since the images are natural RGB images, and binarization degrades the quality of the input images greatly, naturally making the reconstructions poorer.

## 5.4 MEAN FILTERING

Similar to binarization, mean filtering is expected to be another good approach of reducing the effect of noise from the adversarial images. Fig.5.1-5.4 show the adversarial images after mean filtering, and their reconstructions. The radius of the filter used is 2.

### 5.4.1 Results

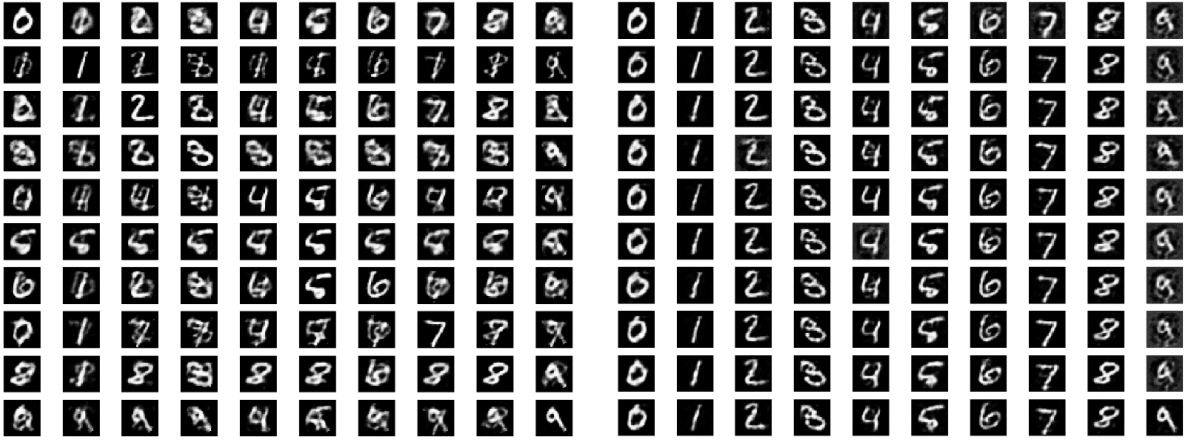
#### MNIST

Tables 5.1 and 5.2 show the metrics after Mean filtering for MNIST. It decreases the target prediction to about 1% and increases the true label prediction to about 94% for attacks against Autoencoder. For VAE, it increases true label prediction to 77%, 57.8% for FGSM and CW attacks, respectively. Visually, it is a strong defense like binarization.

#### CIFAR-10

Tables 5.3 and 5.4 summarize the result of mean filtering. The target label prediction drops to about 10%. The true label prediction varies from 17.8% to 38.9% for different attacks and models. It performs better than binarization for both the attacks and models for CIFAR-10. Fig. 5.8 and 5.10 show that the reconstructions for the mean filtered adversarial images generated by CW attack closely resemble the source images. However, for FGSM attack on Autoencoder Fig 5.5 shows that the reconstructions of the mean filtered adversarial images are still noisy. For FGSM attack on VAE (Fig 5.6), the reconstructions are too blurry, but they resemble the source images.

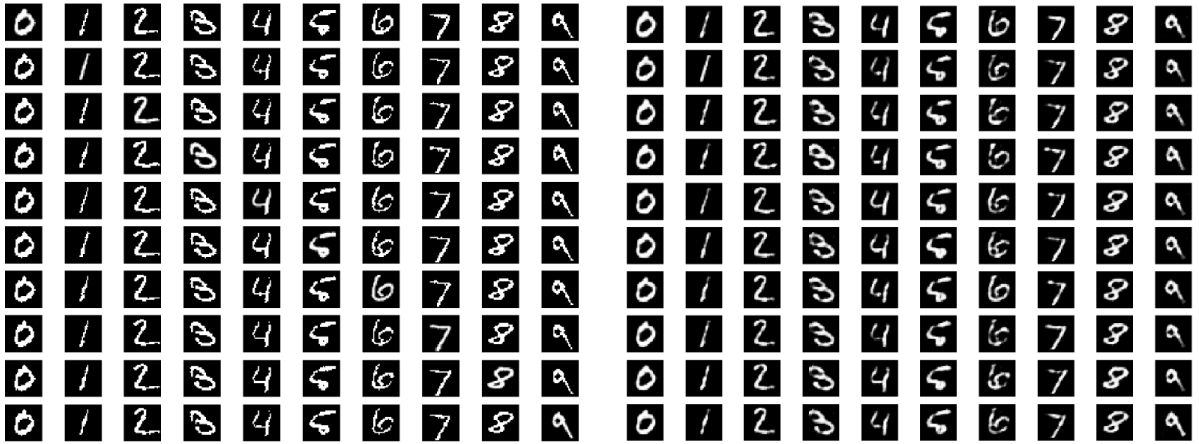




Adversarial Images

Reconstruction of Adversarial Images

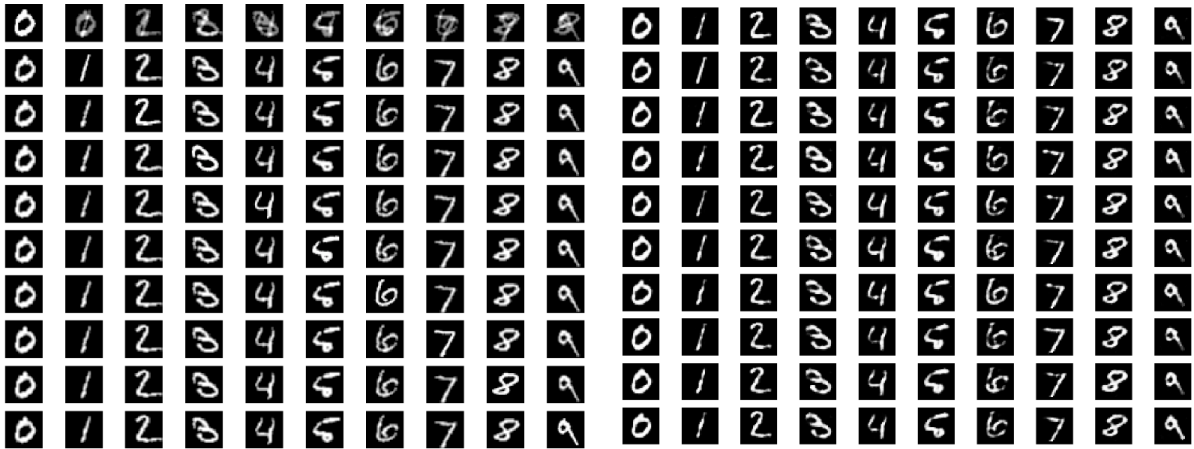
#### Adversarial Training



Adversarial Images

Reconstruction of Adversarial Images

#### Binarization of Adversarial Examples

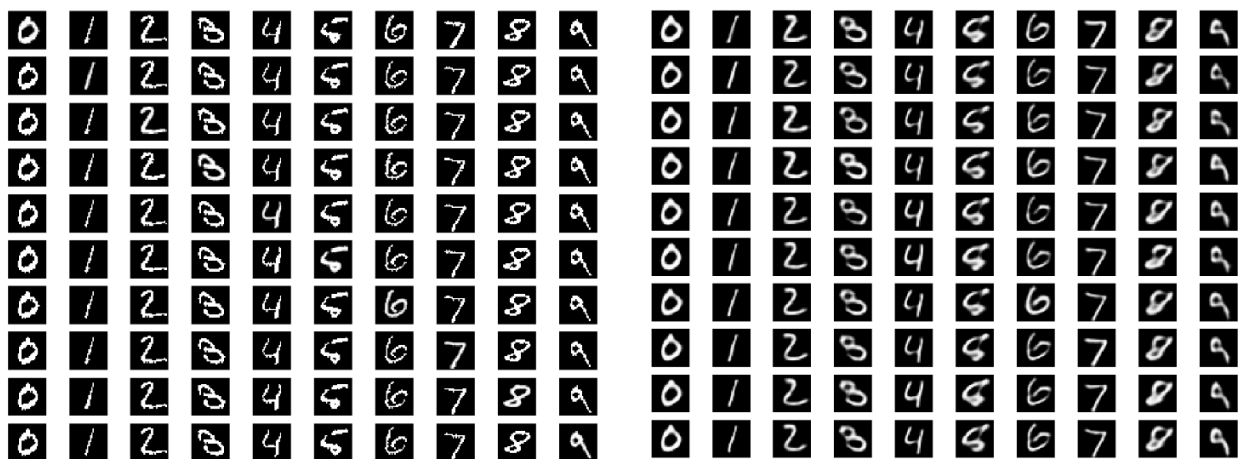


Adversarial Images

Reconstruction of Adversarial Images

#### Mean-filtering of Adversarial Examples

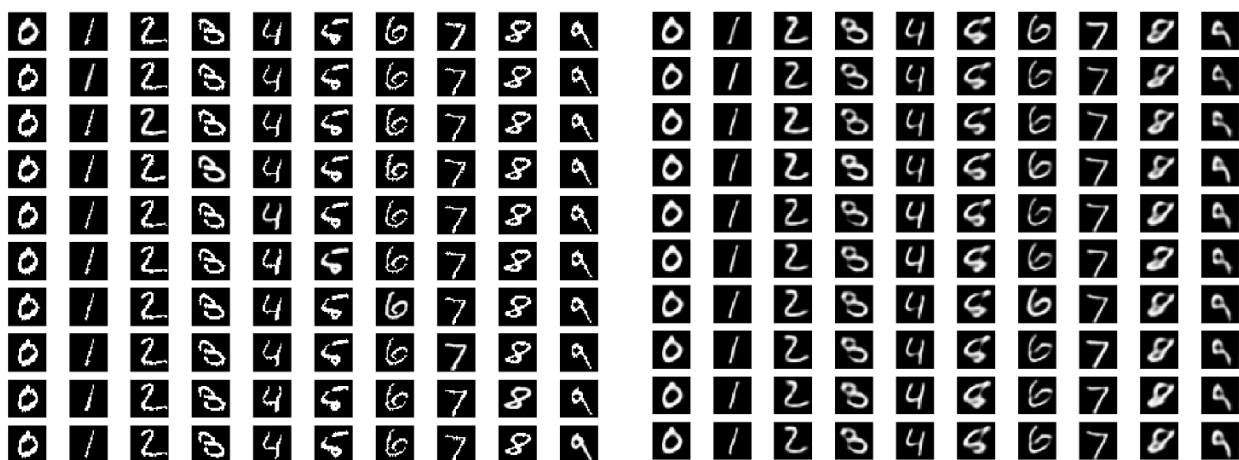
Figure 5.1: Defenses for FGSM on Autoencoder: Adversarial Training, Binarization, Mean-Filtering



Adversarial Images

Reconstruction of Adversarial Images

Binarization of Adversarial Examples

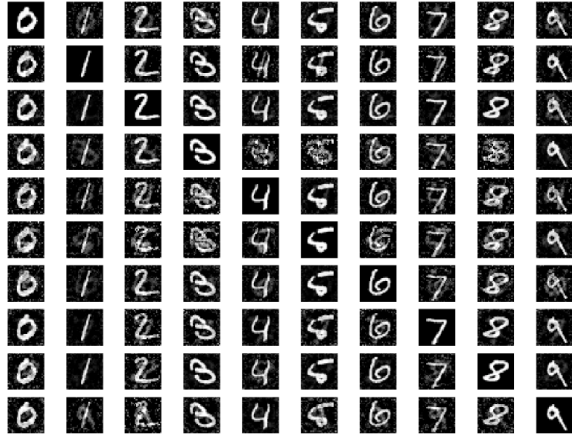


Adversarial Images

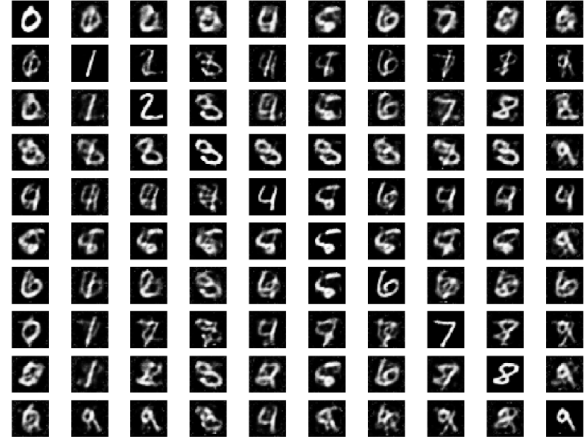
Reconstruction of Adversarial Images

Mean-filtering of Adversarial Examples

Figure 5.2: Defenses for FGSM on VAE: Adversarial Training, Binarization, Mean-Filtering

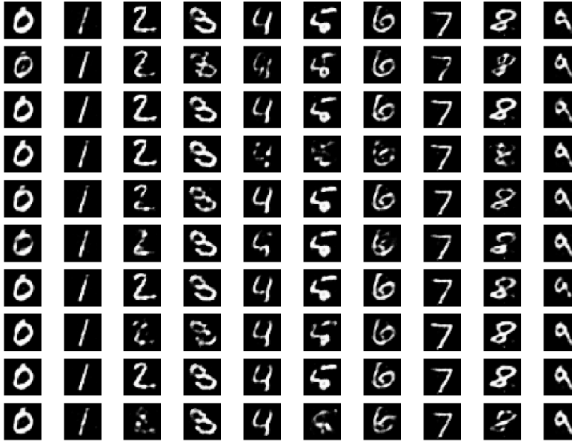


Adversarial Images

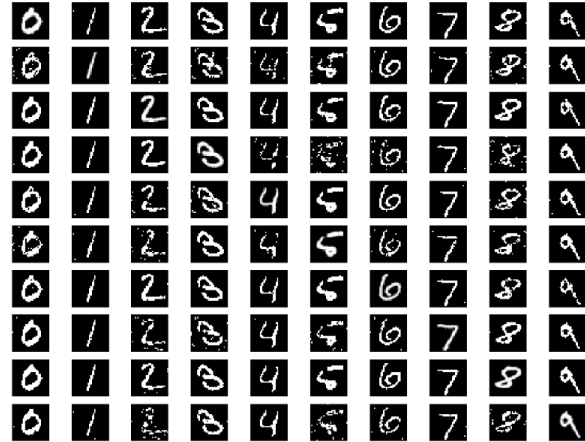


Reconstruction of Adversarial Images

### Adversarial Training

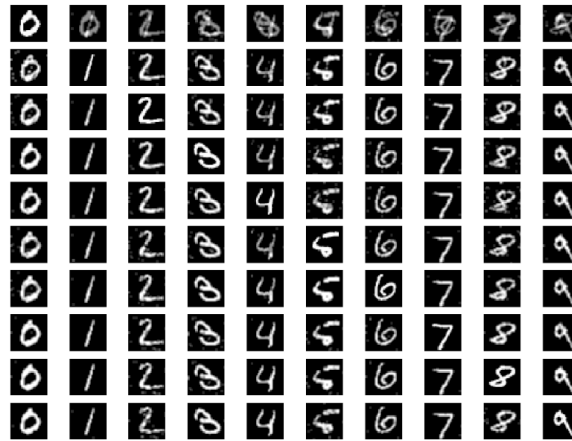


Adversarial Images

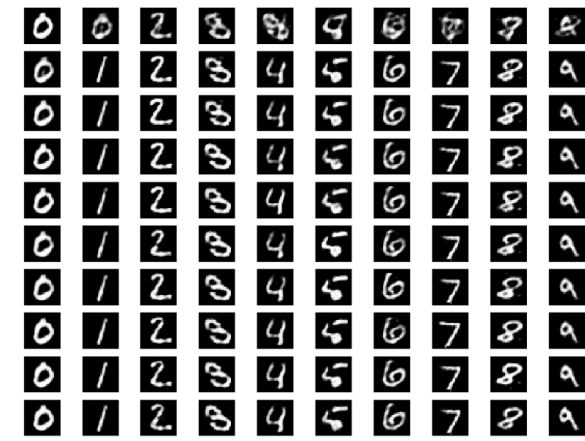


Reconstruction of Adversarial Images

### Binarization of Adversarial Examples



Adversarial Images

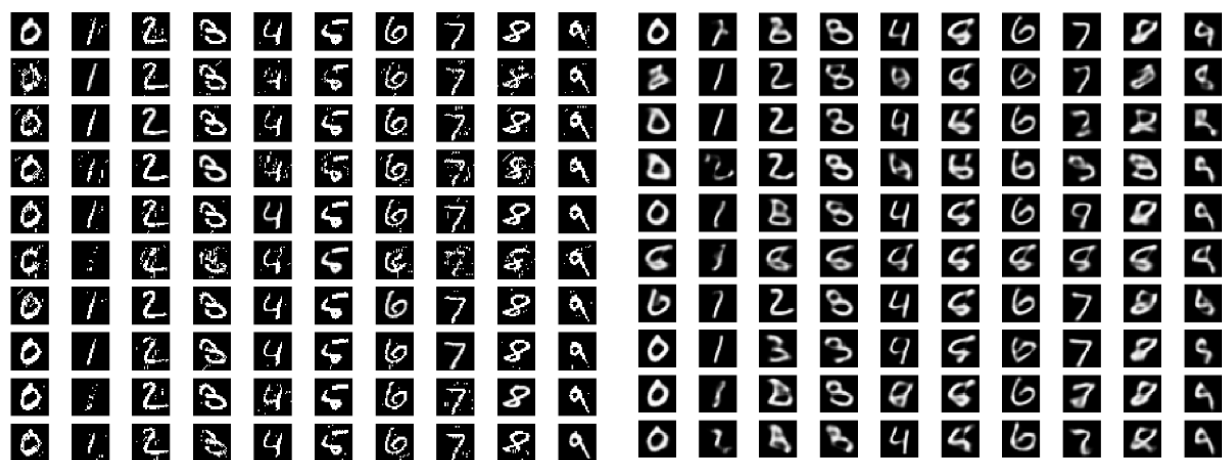


Reconstruction of Adversarial Images

### Mean-filtering of Adversarial Examples

Figure 5.3: Defenses for CW on Autoencoder: Adversarial Training, Binarization, Mean-Filtering

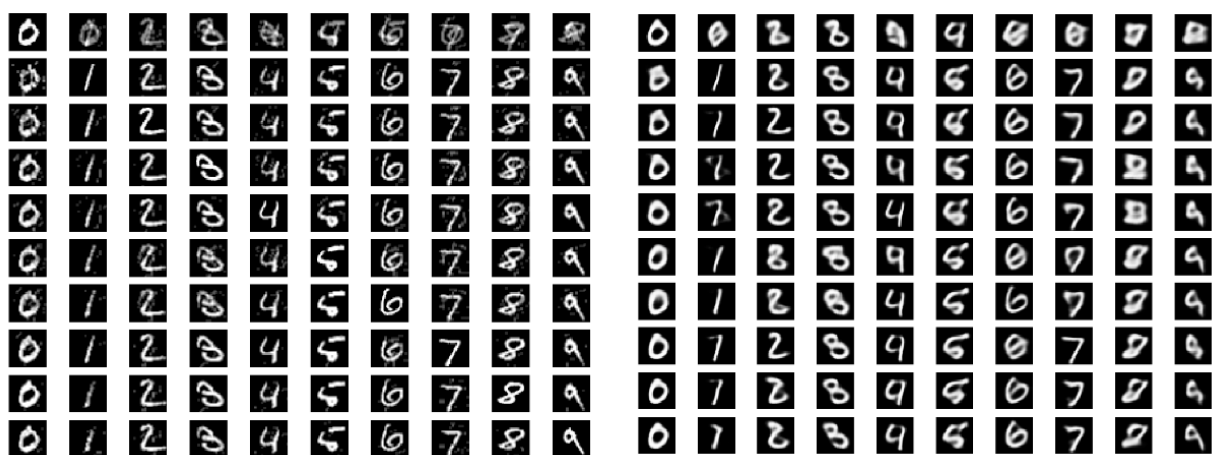




Adversarial Images

Reconstruction of Adversarial Images

Binarization of Adversarial Examples



Adversarial Images

Reconstruction of Adversarial Images

Mean-filtering of Adversarial Examples

Figure 5.4: Defenses for CW on VAE: Binarization, Mean-Filtering

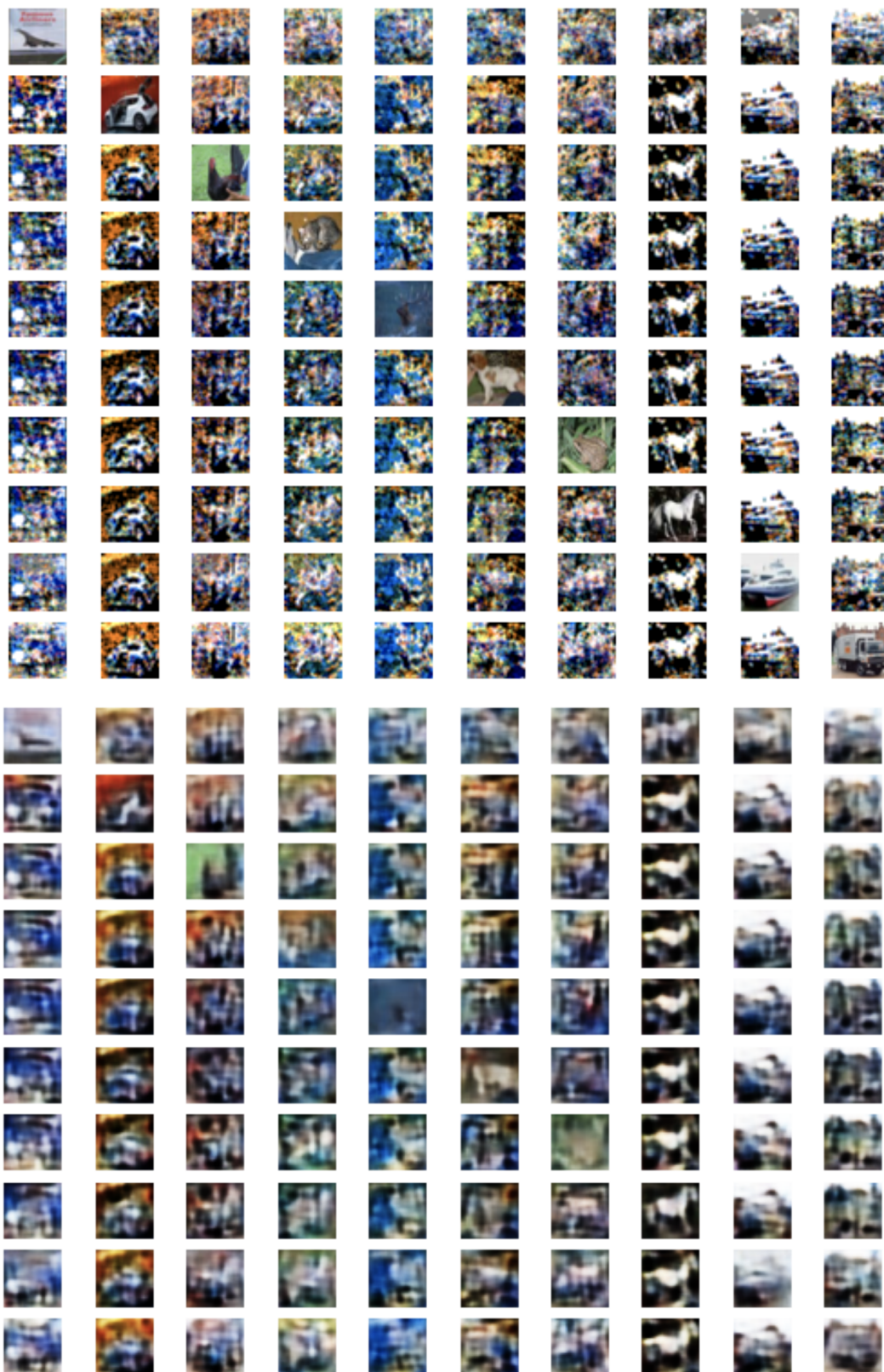


Figure 5.5: Mean filtering Defense for FGSM attack on CIFAR-10, Autoencoder Model: Mean filtered adversarial Inputs (top) and their reconstructions(bottom)

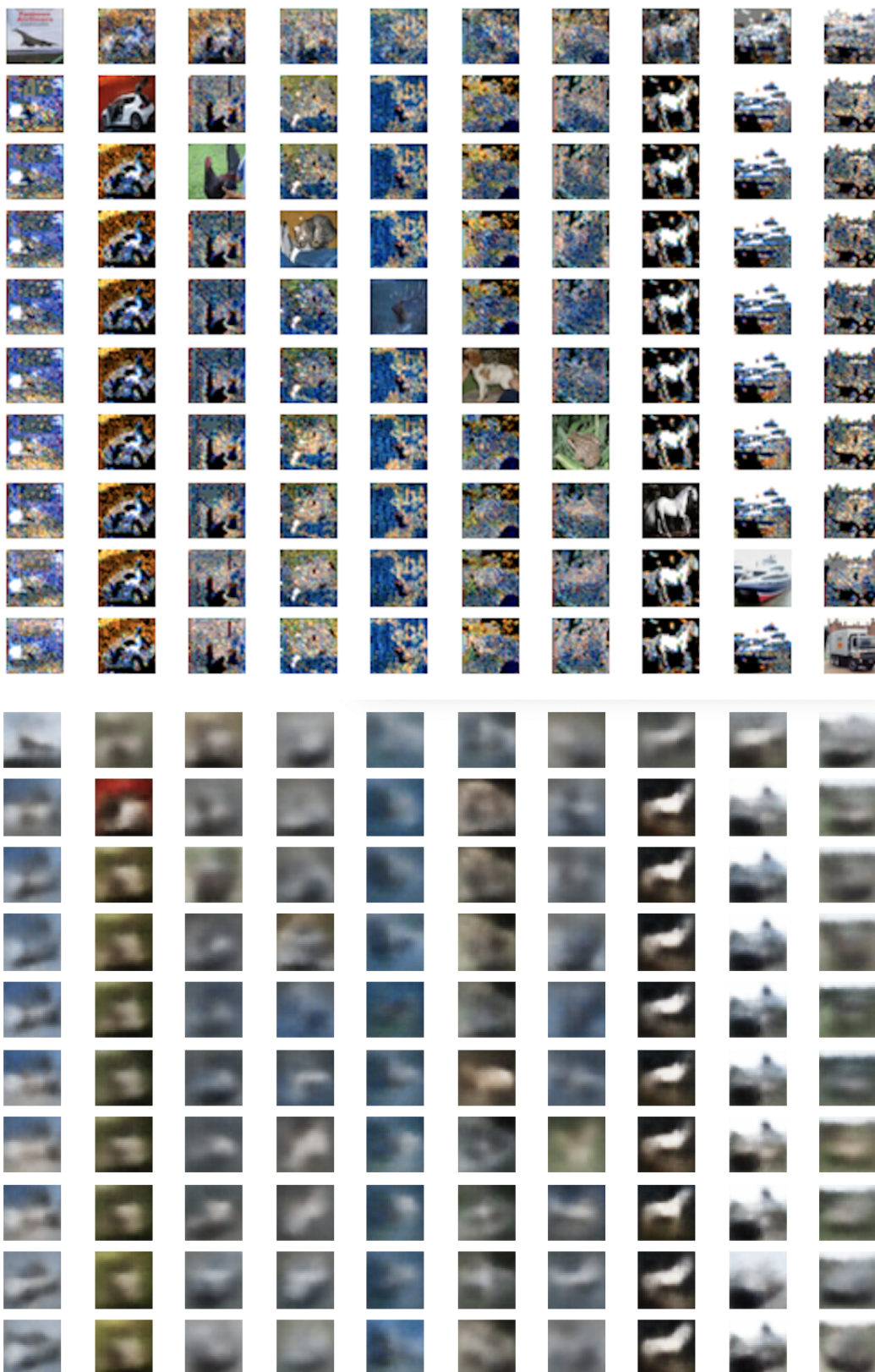


Figure 5.6: Mean filtering Defense for FGSM attack on CIFAR-10, VAE Model: Mean filtered adversarial Inputs (top) and their reconstructions(bottom)



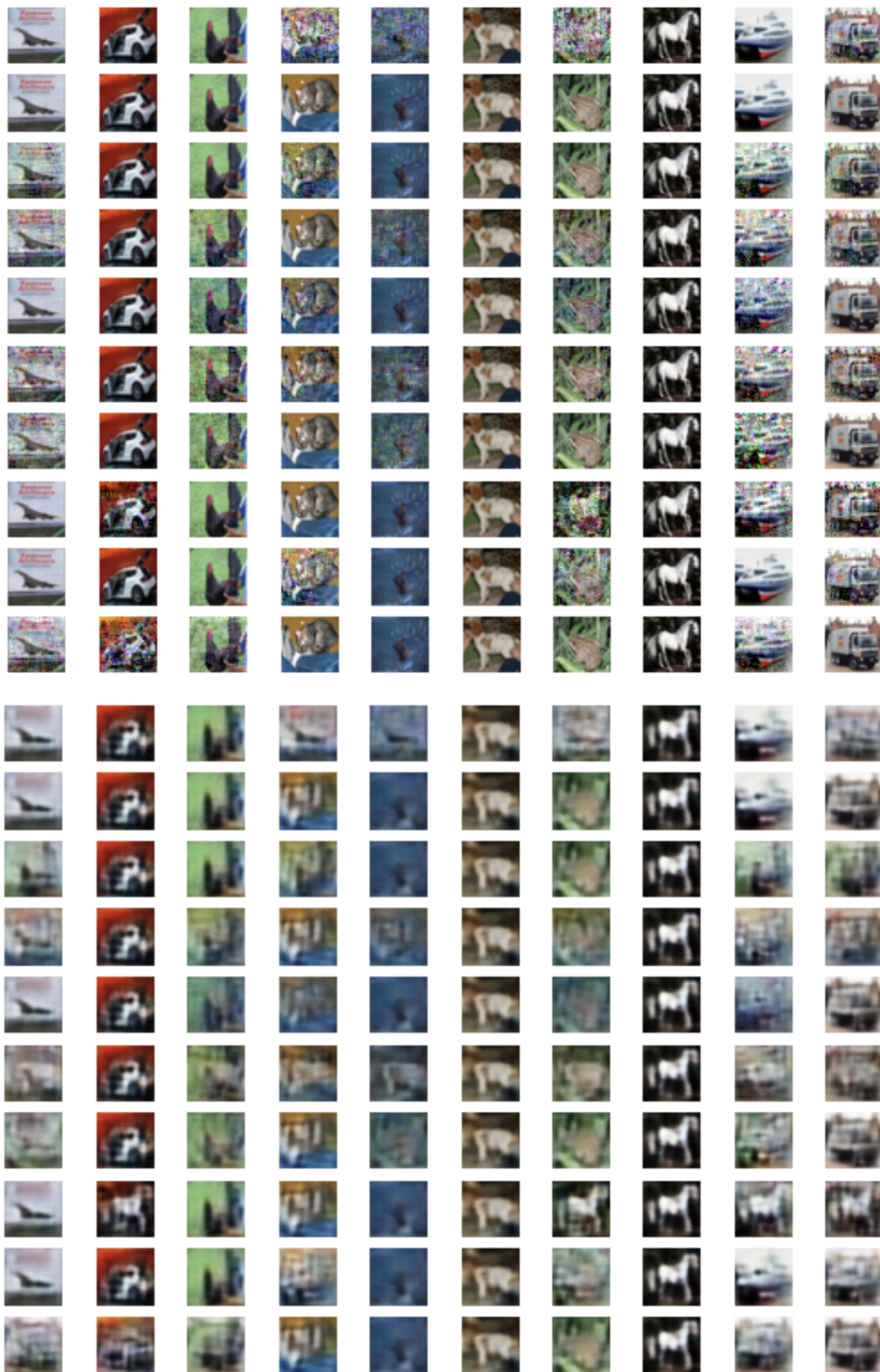


Figure 5.7: Adversarial Training Defense for CW attacks on CIFAR-10, Autoencoder Model: Adversarial Inputs (top) and their reconstructions(bottom)

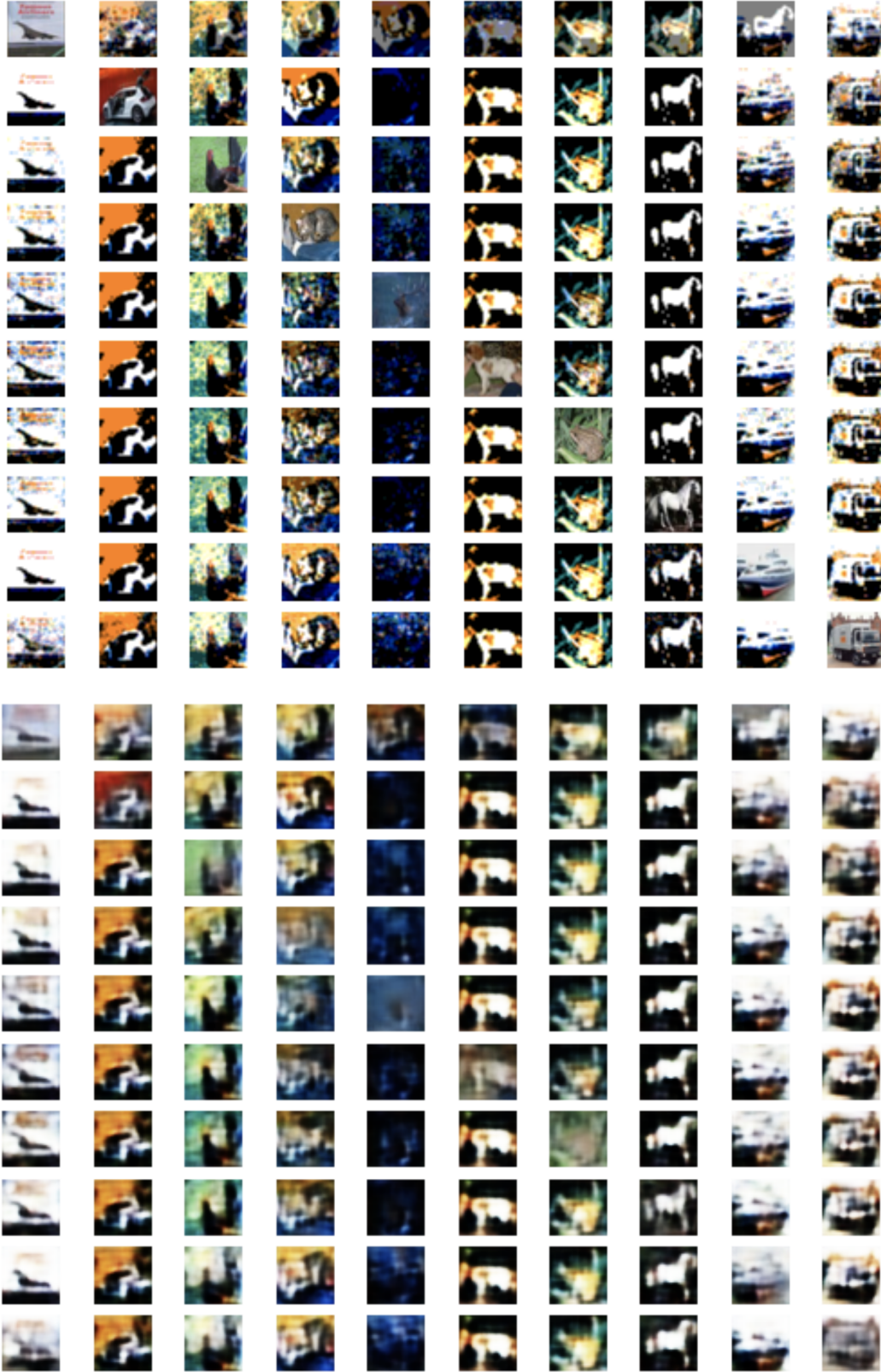


Figure 5.8: Mean Filtering Defense for CW attacks on CIFAR-10, Autoencoder Model: Mean filtered adversarial inputs (top) and their reconstructions(bottom)



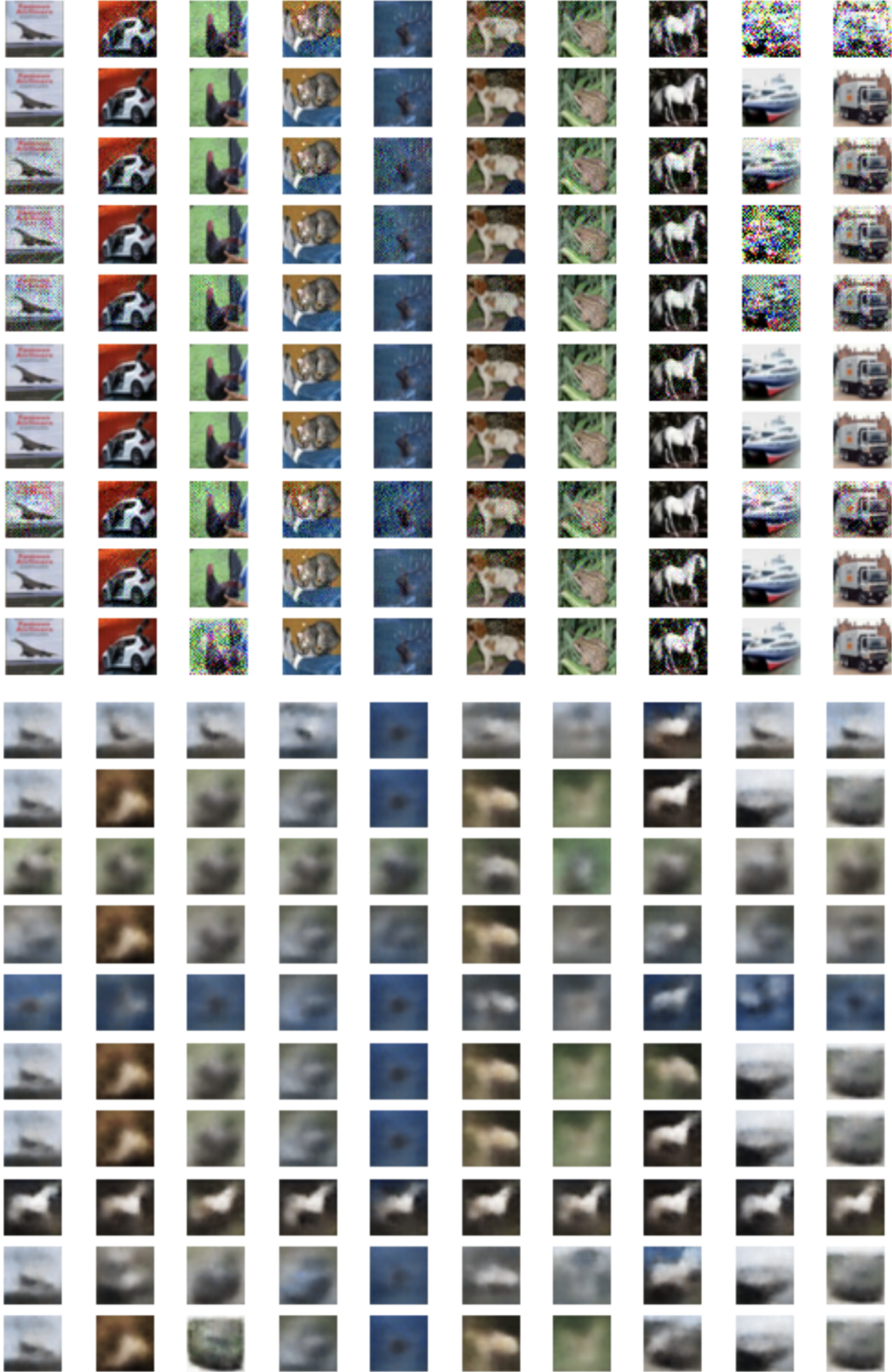


Figure 5.9: Adversarial Training Defense for CW attacks on CIFAR-10, VAE Model: Adversarial inputs (top) and their reconstructions(bottom)

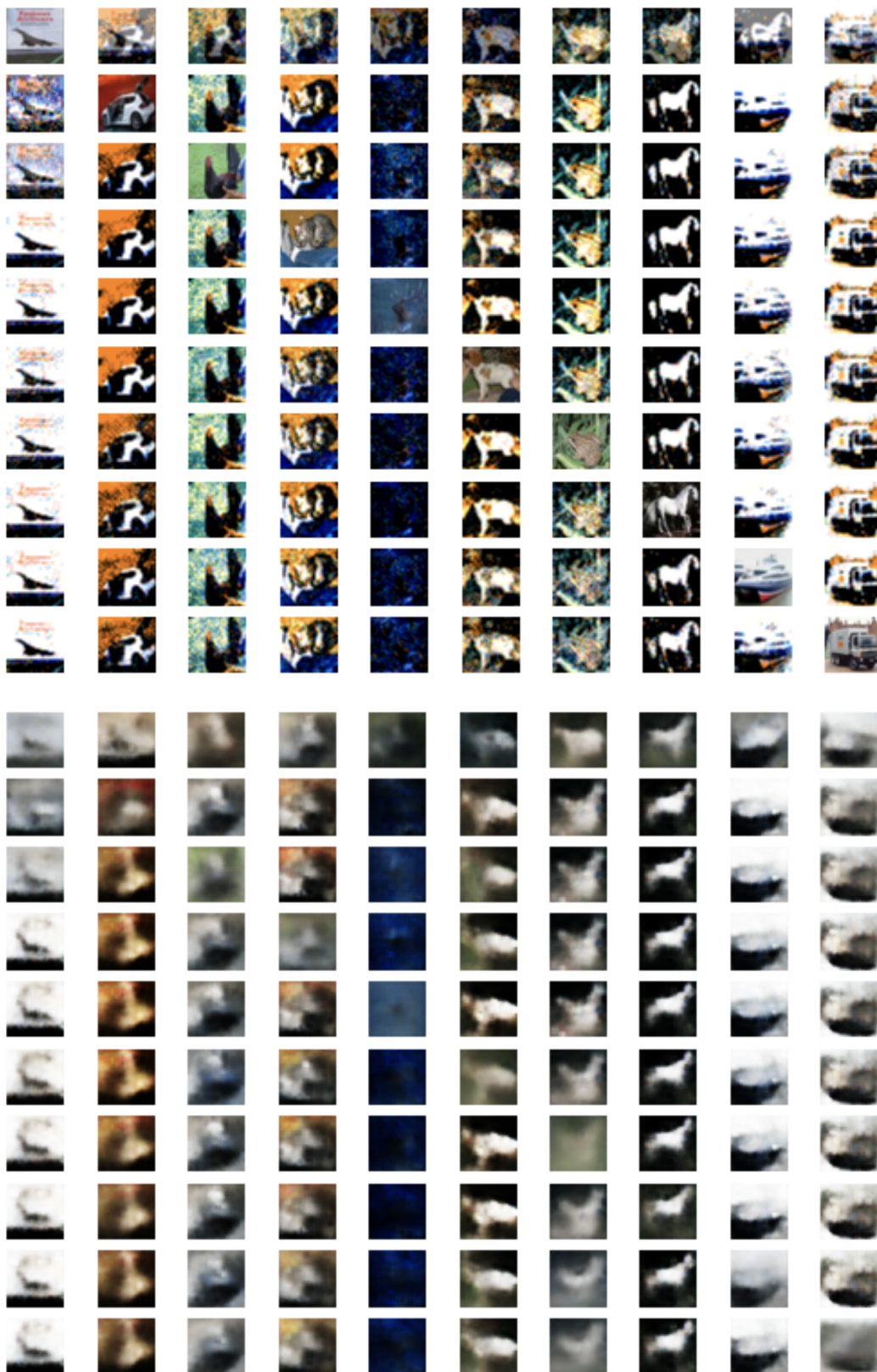


Figure 5.10: Mean Filtering Defense for CW attacks on CIFAR-10, VAE Model: Mean filtered adversarial inputs (top) and their reconstructions(bottom)

ATTACK	DEFENSE	METRIC	
FGSM	Adversarial Training	d1	3.98
		d2	10.14
		noise	2.98
		class_acc_clean	0.97
		class_acc_adv_target	0.056
		class_acc_adv_true	0.785
	Binarization	d1	13.17
		d2	88.09
		noise	4.53
		class_acc_clean	0.97
		class_acc_adv_target	0.011
		class_acc_adv_true	0.922
	Mean Filtering	d1	24.39
		d2	93.96
		noise	4.53
		class_acc_clean	0.97
		class_acc_adv_target	0.01
		class_acc_adv_true	0.944
CW	Adversarial Training	d1	39.9
		d2	36.13
		noise	3.71
		class_acc_clean	0.97
		class_acc_adv_target	1.0
		class_acc_adv_true	0.0
	Binarization	d1	16.62
		d2	85.96
		noise	4.28
		class_acc_clean	0.91
		class_acc_adv_target	0.01
		class_acc_adv_true	0.91
	Mean Filtering	d1	23.78
		d2	95.24
		noise	4.56
		class_acc_clean	0.97
		class_acc_adv_target	0.01
		class_acc_adv_true	0.94

Table 5.1: Results for MNIST, Autoencoder



ATTACK	DEFENSE	METRIC	
FGSM	Binarization	d1	19.13
		d2	87.37
		noise	3.79
		class_acc_clean	0.97
		class_acc_adv_target	0.011
		class_acc_adv_true	0.855
	Mean Filtering	d1	30.32
		d2	91.20
		noise	4.53
		class_acc_clean	0.97
		class_acc_adv_target	0.033
		class_acc_adv_true	0.77
CW	Binarization	d1	34.76
		d2	71.63
		noise	4.79
		class_acc_clean	0.97
		class_acc_adv_target	0.1
		class_acc_adv_true	0.633
	Mean Filtering	d1	36.11
		d2	90.36
		noise	4.71
		class_acc_clean	0.97
		class_acc_adv_target	0.022
		class_acc_adv_true	0.578

Table 5.2: Results for MNIST, VAE

ATTACK	DEFENSE	METRIC	
FGSM	Binarization	d1	295.65
		d2	280.26
		noise	26.09
		class_acc_clean	0.72
		class_acc_adv_target	0.122
		class_acc_adv_true	0.133
	Mean Filtering	d1	204.86
		d2	275.37
		noise	15.89
		class_acc_clean	0.72
		class_acc_adv_target	0.11
		class_acc_adv_true	0.178
CW	Adversarial Training	d1	57.78
		d2	257.07
		noise	5.29
		class_acc_clean	0.72
		class_acc_adv_target	0.344
		class_acc_adv_true	0.2
	Binarization	d1	176.43
		d2	543.62
		noise	18.43
		class_acc_clean	0.72
		class_acc_adv_target	0.122
		class_acc_adv_true	0.144
	Mean Filtering	d1	135.29
		d2	504.54
		noise	12.95
		class_acc_clean	0.72
		class_acc_adv_target	0.078
		class_acc_adv_true	0.389

Table 5.3: Results for CIFAR-10, Autoencoder

ATTACK	DEFENSE	METRIC	
FGSM	Binarization	d1	702.87
		d2	447.94
		noise	25.17
		class_acc_clean	0.72
		class_acc_adv_target	0.178
		class_acc_adv_true	0.044
	Mean Filtering	d1	128.6
		d2	252.76
		noise	11.956
		class_acc_clean	0.72
		class_acc_adv_target	0.11
		class_acc_adv_true	0.244
CW	Adversarial Training	d1	138.87
		d2	197.27
		noise	7.4
		class_acc_clean	0.72
		class_acc_adv_target	0.533
		class_acc_adv_true	0.133
	Binarization	d1	288.18
		d2	434.97
		noise	18.57
		class_acc_clean	0.72
		class_acc_adv_target	0.2
		class_acc_adv_true	0.189
	Mean Filtering	d1	133.74
		d2	467.24
		noise	11.54
		class_acc_clean	0.72
		class_acc_adv_target	0.055
		class_acc_adv_true	0.267

Table 5.4: Results for CIFAR-10, VAE

## 5.5 DISCUSSION

We implemented L-BFGS, FGSM and Carlini Wagner iterative  $L_2$  attack on Autoencoder and VAE models trained on MNIST and CIFAR-10 datasets. L-BFGS attack proved to be the strongest attack on MNIST, however due to its computational inefficiency, it is not feasible for attacking more complex datasets like CIFAR-10. FGSM was not very effective in targeted attack, however it did perturb the reconstructions such that they did not resemble the original images for MNIST. Carlini-Wagner  $L_2$  iterative attack was effective against MNIST autoencoder and VAE, and CIFAR-10 VAE. Also, VAE was in general easier to attack than autoencoders, probably because the VAE was trained for very few epochs. Of the defenses, Mean filtering was the most effective out of the three defenses discussed. It was an interesting result, since it is such a simple pre-processing step, yet it makes the models drastically robust to the attacks, while intensive methods like Adversarial training provided little or no benefit.

All the attack and defense methods were built in `cleverhans`. The code is publicly available on `github` and we hope it would be of help to other researchers interested in exploring this space further.

## 5.6 FUTURE WORK

We implemented two simple attacks in this work on Autoencoders and VAEs. The reconstructions by VAE were quite blurry even for the clean images, so we need to analyze these attacks on a more sophisticated model like VAE-GAN, which produces sharper reconstructions. The defense methods studied here were rather simple, involving adversarial training or image pre-processing. It would be worthwhile to explore more advanced defense methods, like defensive distillation or JPEG compression as a pre-processing step. Further, if the adversary is aware of the defense mechanism being used, they may be able to generate better attacks than they can without that knowledge. For instance, for image pre-processing defense, the adversary would generate attacks by backpropagating through the entire network including the pre-processing layer. We expect such an attack to be effective against the mean filtering defense, however it might be tough to backpropagate in the case of a binary threshold filter, since the adversary would need to solve a discrete optimization problem. Another important defense method which we did not explore was adversary detection. One can look at the latent embedding of the original image, and compare it with the latent embedding of its reconstruction. We expect that for a clean image, these two embeddings would be more

similar, than for an adversarial image. We tested this on FGSM attack, and although it was true when the encoder of the Autoencoder was used to get the latent embeddings, on using a different encoder, it did not hold true. Thus, if the receiver does not have access to the same encoder as the sender, they will not be able to detect adversarial examples using this approach. It would be worthwhile to test some adversarial detection techniques which have been proven to work for classification models.

link to github repository<sup>1</sup>

---

<sup>1</sup><https://github.com/iirishikaii/cleverhans>

## REFERENCES

- [1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.
- [2] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, “Autoencoding beyond pixels using a learned similarity metric,” *arXiv preprint arXiv:1512.09300*, 2015.
- [3] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.
- [4] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks,” in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 582–597.
- [5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.
- [6] J. Lu, T. Issaranon, and D. Forsyth, “Safetynet: Detecting and rejecting adversarial examples robustly,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 446–454.
- [7] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, “Pixeldefend: Leveraging generative models to understand and defend against adversarial examples,” *arXiv preprint arXiv:1710.10766*, 2017.
- [8] J. Kos, I. Fischer, and D. Song, “Adversarial examples for generative models,” in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 36–42.
- [9] P. Tabacof, J. Tavares, and E. Valle, “Adversarial images for variational autoencoders,” *arXiv preprint arXiv:1612.00155*, 2016.
- [10] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Citeseer, Tech. Rep., 2009.
- [11] 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.