Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2023) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

International Neural Network Society Workshop on Deep Learning Innovations and Applications (INNS DLIA 2023)

Evaluating Robustness of Reconstruction Models with Adversarial Networks

Korn Sooksatra[a,*], Gissella Bejarano[a], Pablo Rivas[a]

[a]*Department of Computer Science, Baylor University, 1311 S 5th St, Waco, TX 76706*

Abstract

With the advent of adversarial robustness as a research area, much novel work attempts to design creative defense mechanisms against adversarial vulnerabilities that arise. While classification models are the most common target of adversarial robustness research, reconstruction models are often underestimated though they play essential roles in many applications. This work evaluates reconstruction models regarding their adversarial robustness. We constructed two frameworks: a standard and a universal-attack framework. The standard framework requires an input to find its perturbation, and the universal-attack framework generates adversarial perturbation from the distribution of a dataset. Extensive experimental evidence discussed in this paper suggests that both frameworks can effectively alter how images are reconstructed and classified using classic reconstruction models trained on MNIST and Cropped Yale Face datasets. Further, these frameworks outperform state-of-the-art adversarial attacks. Moreover, we showcase using the proposed framework to retrain a reconstruction model to improve its resilience against adversarial perturbations. Furthermore, for the sake of reconstruction models, an attack may desire not to alter the latent space. Thus, we also include the analysis of the latent space.

Keywords: adversarial robustness; reconstruction models; adversarial vulnerabilities; adversarial attacks; latent space analysis

1. Introduction

The societal impact of machine learning applications is self-evident as it continues to solve many highly complex tasks (e.g., self-driving cars [1]). However, many of these models are still susceptible to small perturbations that naturally occur in the world or are fabricated by learning to model the input space. In the latter case, models could be trained to be robust against noise. Goodfellow *et al.* [2] and Szegedy *et al.* [3] demonstrated how to create small additive perturbations to mislead classification models and expose the issue. Their research shows that many state-of-the-art models were not robust against a perturbation that is currently known as an *adversarial example*. Naturally,

* Korn Sooksatra
E-mail address: korn_sooksatra1@baylor.edu

many works [2, 4, 5, 6] started trying to determine how to defend models against adversarial examples effectively. While defense mechanisms were developed, others worked around this finding new adversarial examples against such defenses [7, 8, 9, 10]. Note, however, that most of the existing work focuses on adversarial examples for classification models, even though reconstruction models are widely used in practice (e.g., people in clothing [11], music generation [12], molecule graphs generation [13] and image compression). Therefore, our work attempts to construct frameworks for evaluating the adversarial robustness of reconstruction models. In particular, we focus on models whose goal is to reconstruct the input. Adversarial examples in this setup try to make the targeted models generate an output that belongs to a different class than the corresponding input. We use the idea of well-known GANs [14] to train a generator to produce small perturbations. Then, we add such perturbations to an image to create an adversarial example for a targeted reconstruction model. This paper introduces two frameworks to demonstrate the general idea, i.e., a standard and a universal-attack framework. The standard framework trains a generator such that it requires input to produce perturbation during execution. On the contrary, the universal-attack framework trains a generator that does not need an input to produce a perturbation during execution. The proposed methodology produces generators effective on MNIST and Cropped Yale Face dataset (CYFD).

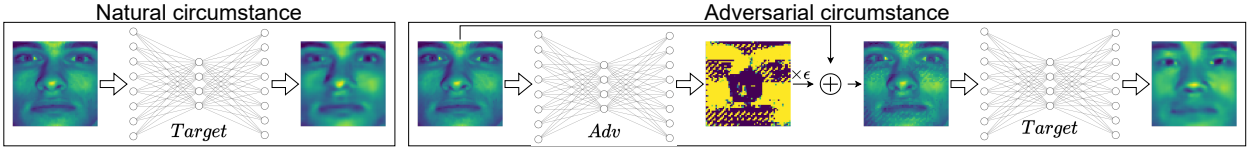


Fig. 1. Example of our trained generator misleading a targeted reconstruction model where *Target* is the targeted model and *Adv* is our generator. This shows what happens in a natural scenario (left) and an adversarial scenario (right).

Fig. 1 shows how to apply our trained generator to mislead a target. The natural circumstance (left) shows that the target works fine with a clean image since the reconstructed image looks similar to the clean image. However, a clean image fed to our trained generator in the adversarial circumstance (right) produces a perturbation multiplied by ϵ (i.e., the perturbation bound). Then, it is added to the clean image to create an adversarial example. Note that when the adversarial example enters the target model, the output is a face from a completely different class than the input. We can retrain the target models with our framework to make them robust against these perturbations. Moreover, we analyze the latent spaces of the target models and found an interesting pattern. That is, the differences between latent spaces of the clean samples and their corresponding adversarial examples increase when the accuracy on the reconstructed images of the adversarial examples increases in MNIST. However, we did not find this pattern in CYFD. This paper showcases the positive effects of such retraining on MNIST and CYFD data with good results.

Our main contributions can be summarized as follows:

- We formally define a data reconstruction problem based on reconstruction models introducing adversarial examples.
- We design and implement a standard and universal-attack framework to train adversaries for reconstruction models, producing effective adversarial examples using the well-known approach by [14].
- We successfully use the proposed standard framework to retrain a reconstruction model improving its adversarial robustness.

2. Related Works

Because our framework applies GANs to evaluate reconstruction models, this section briefly describes previous works that applied GANs to create adversarial examples for discriminative models in Section 2.1. We further go over existing works creating adversarial examples for reconstruction models by using optimization-based attacks in Section 2.2.

2.1. GANs-Based Attacks

Several works apply GANs to create adversarial examples in classification models. However, the most related works are described as follows. Xiao *et al.* [15] utilized GANs to train a generator to create perturbation that can be added to a clean image to generate an adversarial example for a classification model. They proposed the attack under both semi-whitebox and blackbox settings. Bai *et. al* [16] applied the work of [15] and proposed a solution to make it converge earlier during the training. Their solution is to train a generator with the existing adversarial examples in the first stage to make the generator know the direction to create perturbation in the same space as one of the adversarial examples. In the second stage, they followed the training in [15]. Intuitively, the existing works in this area do not consider the reconstruction models.

2.2. Adversarial Examples on Reconstruction Models

Only a few works focused on finding adversarial examples in reconstruction models. Tabacof *et al.* [17] tried to find an adversarial example of a clean image by minimizing the difference between the adversarial example's latent space and the target's image latent space where the perturbation added to the clean image was minimized. Kos *at al.* [18] proposed three attacks against reconstruction models: classifier attack, $\mathcal{L}_{VAE}$ attack and latent attack. They were all based on optimization problems and used Carlini and Wagner attack [10] to approximate the solutions. So far, there are only attacks based on optimization problems that take time to approximate a solution during testing. In 2020, Pope *et al.* [19] applied the projected gradient descent [6] for evaluating reconstruction models.

To the best of our knowledge, our work is the first to adopt GANs to find adversarial examples of reconstruction models.

3. Our Approach

This section explains our attacks in detail and starts with defining the problem of adversarial examples in reconstruction models. Then, we derive the loss functions of the standard framework and describe how to train the generator with the framework. Furthermore, we discuss the universal-attack framework.

3.1. Problem Definition

Suppose that there is a reconstruction model T (e.g., a VAE) trained on clean dataset $\mathcal{X} \subseteq \mathbb{R}^n$, where n is the number of features. We also denote a class set as $\mathcal{Y}$ and instance i sampled from distribution P as (x_i, y_i) where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$. Note that y_i is the ground truth class of x_i. The goal of an adversary is to create an adversarial example $(x_i^{'})$ from x_i such that $x_i^{'}$ looks like samples in class y_i; however, $T(x_i^{'})$ belongs to another class (not equal to y_i). Our approach generates perturbation which is bound by L_∞ and then adds it to a clean input to create an adversarial example. Note that $L_\infty(x_i, x_i^{'}) = \max|x_i - x_i^{'}|$.

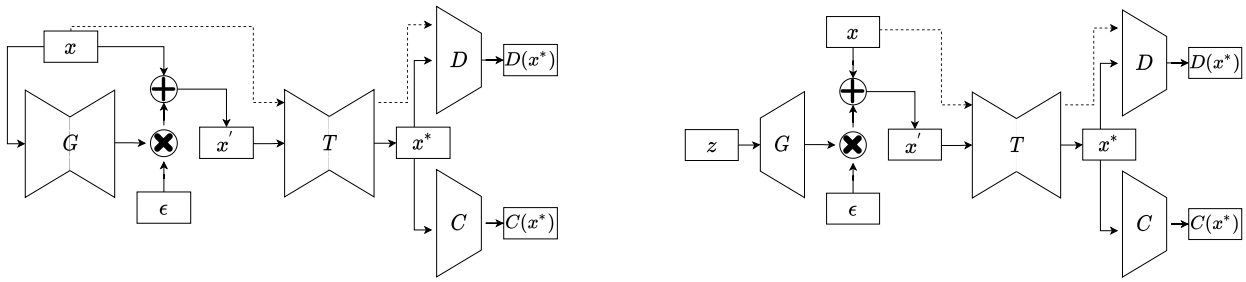
3.2. Standard Framework (SF)

The main concept of our framework is to train a generator to create perturbation such that this perturbation can be added to clean input and generate an adversarial example for a target network (T). Fig. 2a shows our framework consisting of a generator (G), a discriminator (D) and a pretrained classifier (C). In the following, we assume that x is an arbitrary member of $\mathcal{X}$. Each component in the framework has its own role in the framework as follows. D is used to remain x^* in the same distribution of $T(x)$. To achieve it, D needs to minimize

$$L_D = \log(D(T(x)) + \log(1 - D(x^*)), \tag{1}$$

where $T(x^{'}) = x^*$. Also, G needs to minimize

$$L_G = \log(D(x^*)). \tag{2}$$



(a) Standard framework

(b) Universal-attack framework

Fig. 2. The architecture of the standard and universal-attack frameworks where G is the generator, D is the discriminator, C is the pretrained classifier, T is the targeted reconstruction model, x is an instance from $\mathcal{X}$, z is a random noise from the normal distribution and ϵ is the perturbation bound.

As a result, D should not be able to distinguish between $T(x)$ and x^*. Further, C directs G to update its parameters such that $C(x^*) \neq y$ where $C(x^*)$ is a class classified by C given x^*, and y is the ground truth label of x. Therefore, the loss regarding C is the opposite of categorical cross entropy used for training a classification network, and we can define it as

$$L_C = \sum_{j=0}^{|\mathcal{Y}|-1} l[j] \log(Z(x^*)[j]), \tag{3}$$

where l is the ground truth logit, $l[j]$ is the value at index j of l and $Z(x^*)[j]$ is the logit at class j of C given x^* (i.e., the output before softmax activation). Note that if $\mathcal{X}$ is single-class data, l has 1 only in one index. Nevertheless, if $\mathcal{X}$ is multi-class data, l can have 1 in multiple indexes. Moreover, we also want to maximize the difference between $x^{'}$ and x^* so that $x^{'}$ is not similar to x^*. We use the opposite of mean square error as

$$L_{MSE} = -\sum_{j=0}^{n-1} (x^{'}[j] - x^*[j])^2, \tag{4}$$

where $x^{'}[j]$ is feature j of $x^{'}$. Therefore, the total loss of G is

$$L_{Total} = L_{MSE} + \lambda_G L_G + \lambda_C L_C, \tag{5}$$

where λ_G and λ_C are hyperparameters to balance all the losses. Also, ϵ is the perturbation bound, and we use L_∞ as described earlier.

During the training time, the pretrained classifier (C) and the targeted reconstruction model (T) are untrainable. Then, the framework processes as follows: a) feed x to G and obtain perturbation; b) multiply the perturbation with ϵ to fit the perturbation in the bound and add it to x to create $x^{'}$; c) feed $x^{'}$ to target T and obtain x^*; d) feed x to T and obtain $T(x)$; e) feed x^* and $T(x)$ to D to compute L_D and L_G and feed x^* to C to compute L_C f) compute L_{MSE} from $x^{'}$ and x^*; g) compute L_{Total} from the three losses and update the parameters of D and G by using a gradient descent method on L_D and L_{Total} respectively.

We can only feed an arbitrary input to the generator (G) during the testing time and obtain perturbation. Then, we multiply the perturbation with ϵ and add it to the input. Consequently, we obtain an adversarial example of the input. Note that the last layer of G has to be a *tanh* activation function to limit the adversarial example in the bound ϵ.

Noticeably, this framework is an untargeted attack because it tries to find an adversarial example ($x^{'}$) such that $C(x^*) \neq y$. Hence, T can produce an instance belonging to any class except the correct class. Additionally, we can also adjust it to create a targeted attack as follows. Instead of using the logit of the ground-truth label in (3), we use the logit of the targeted label instead. Also, we use the categorical cross-entropy loss instead for L_C. Thus, the loss is

$$\sum_{j=0}^{|\mathcal{Y}|-1} t[j] \log(Z(x^*)[j]), \tag{6}$$

where t is the logit of the target class, and $t[j]$ is the value at index j of t.

3.3. Universal-Attack Framework (UF)

In the real world, some attacks may study only the distribution of a dataset and create adversarial perturbation without requiring inputs. Therefore, we design this framework to evaluate a target model with respect to the attacks described earlier. Therefore, we do not feed input to G during the testing time but feed noise z. Fig. 2b demonstrates the framework. All the components and their roles are the same as in the previous framework. However, the training steps are slightly different.

z is sampled from the normal distribution during the training time and fed to G. Then, the rest of the process is the same as the previous framework. That is, the output of G is multiplied by ϵ and then added to x to create $x^{'}$. Then, $x^{'}$ enters the target model T, and it outputs x^{*}. After that, x^{*} is fed to the discriminator D and the pretrained classifier C. Note that we also perform the same process of $x^{'}$ on x. At last, we use the output from D, C and T to compute gradients and train the framework. As a result, in the testing time, G can generate perturbation later used for creating adversarial examples on samples from the distribution of $\mathcal{X}$ for T.

4. Experiments and Results

In this section, we first evaluate our approach as untargeted and targeted attacks on MNIST [20] and the Cropped Yale Face dataset (CYFD) [21]. MNIST is a hand-written-digit-image dataset consisting of 50000 training samples and 10000 test samples and thus has ten classes. CYFD is a human-face dataset consisting of 1960 training samples and 491 test samples and has 38 classes (i.e., 38 faces). Further, we also explore the universal-attack framework on the same datasets. We choose a variational autoencoder (VAE) [22] and a VAEGAN [23] as our targeted reconstruction models because they are well-known in this kind of model. For different datasets, we use different architectures of components in the frameworks. We used the test samples of the datasets unseen by our attacks to evaluate the efficiency of our attacks. Further, because crowdsourcing was costly, we used the pretrained classifiers in our frameworks to predict classes of the input and output of the targets for the evaluation.

4.1. Architectures and Preprocessing

For MNIST, we used the architecture from [24] for the targeted VAE and the same architecture with a discriminator from DCGAN [25] for the targeted VAEGAN. Further, we applied the architectures of DCGAN [25] for the adversary's generator and discriminator. Note that the encoder part of the generator is the inverse of the decoder part. For the adversary's classifier, we adopted the architecture in [26]. First, we created our targeted models by training the VAE and VAEGAN with the training samples, 20 latent space, 250 batch size, Adam optimizer [27] and 100 epochs. Second, we pretrained the adversary's classifier with the same setting as the targets.

For CYFD, we utilized the architecture from [28] to create the targeted VAE and the same architecture with a discriminator also from [28] to create the targeted VAEGAN. Note that the work in [28] is for generating images of the CelebA dataset [29] which is also a human-face dataset. Moreover, we used the architectures in [28] to create the adversary's generator and discriminator. Note that the encoder part of the generator is the inverse of the decoder part. Then, we applied the architecture of the MNIST classifier to create the adversary's classifier. After that, we trained the VAE and VAEGAN with the training samples, 80 latent space, 245 batch size, Adam optimizer [27] and 100 epochs. At last, we pretrained the classifier with the same setting.

4.2. Baselines

We can use the attack in [19] as a baseline. However, the attack did not use the pretrained classifier, and this pretrained classifier had information to mislead the target model. Thus, it was not fair to use it as a baseline. Then, we have designed a framework on which we can apply state-of-the-art attacks. This framework includes only the pretrained classifier from our approach. First, an input is fed to the target (i.e., VAE or VAE-GAN), and the output is fed into the pretrained classifier. Then, the attack uses the classifier's output to compute the gradients with respect to the input and performs a state-of-the-art gradient-based attack. Further, in our framework, we have two baselines which are based on [19] and our attack. The first one uses the fast gradient sign method (FGSM) [2], and the second one uses projected gradient descent (PGD) [6].

Table 1. Accuracy achieved by the classifier on the sets of samples concerning MNIST and CYFD. Note that adversarial examples were generated by our FGSM, PGD and standard framework (SF).

Model	X	R_X	FGSM		PGD		SF	
			A	R_A	A	R_A	A	R_A
MNIST								
VAE	99.47%	91.51%	98.78%	32.37%	99.39%	34.82%	99.22%	**4.84%**
VAEGAN		94.65%	98.72%	50.37%	99.41%	43.33%	99.15%	**5.63%**
CYFD								
VAE	99.39%	77.8%	70.88%	18.33%	99.98%	37.68%	74.95%	**18.13%**
VAEGAN		82.48%	77.19%	19.14%	99.39%	31.77%	76.37%	**6.92%**

4.3. Standard Framework

For MNIST, we trained our standard framework with the setting of 250 batch size and $\epsilon = 0.1$ for 100 epochs. For CYFD, we trained it with the setting of 140 batch size and $\epsilon = 0.05$. Further, we set λ_G to be 0.5 because most of the images from the targets were valid. For the untargeted attack, we set λ_C to be 1 since we wanted to change the class of the output of the targets. For the targeted attack, we set it to 2 because we would like to focus on changing the class of the reconstructed image, and the reconstructed images did not look like the inputs when increasing it (e.g., 3 and 4).

4.3.1. Untargeted Attack

After we trained the adversary and used our generator to create adversarial examples, then, we denote a set of clean test samples as X, a set of their reconstructed images as R_X, a set of their corresponding generated adversarial examples as A and a set of their reconstructed images as R_A. A is obtained by feeding X to G, multiplying G's output with ϵ and adding it to X. R_A is simply obtained by feeding A to the target model T. Note that we use these notations throughout Section 4 and 5.

Table 1 shows accuracy obtained from the MNIST and CYFD classifiers on those four sets of images according to two baselines and our standard framework (SF). Explicitly, all the attacks (i.e., FGSM, PGD and SF) work very well because the accuracy of the classifiers on the reconstructed images generated from our attack was significantly reduced while the ones on X and A were still high. Specifically, our attack explicitly outperformed the two baselines since the accuracy of the classifier on R_A of SF is significantly lowest. However, in CYFD, PGD did not perform well compared to the other two attacks because it did not find the correct direction to mislead the target. Note that PGD required a gradient direction multiple times. Then, it did not perturb the clean images so much since the classifier's accuracy on A was about the same.

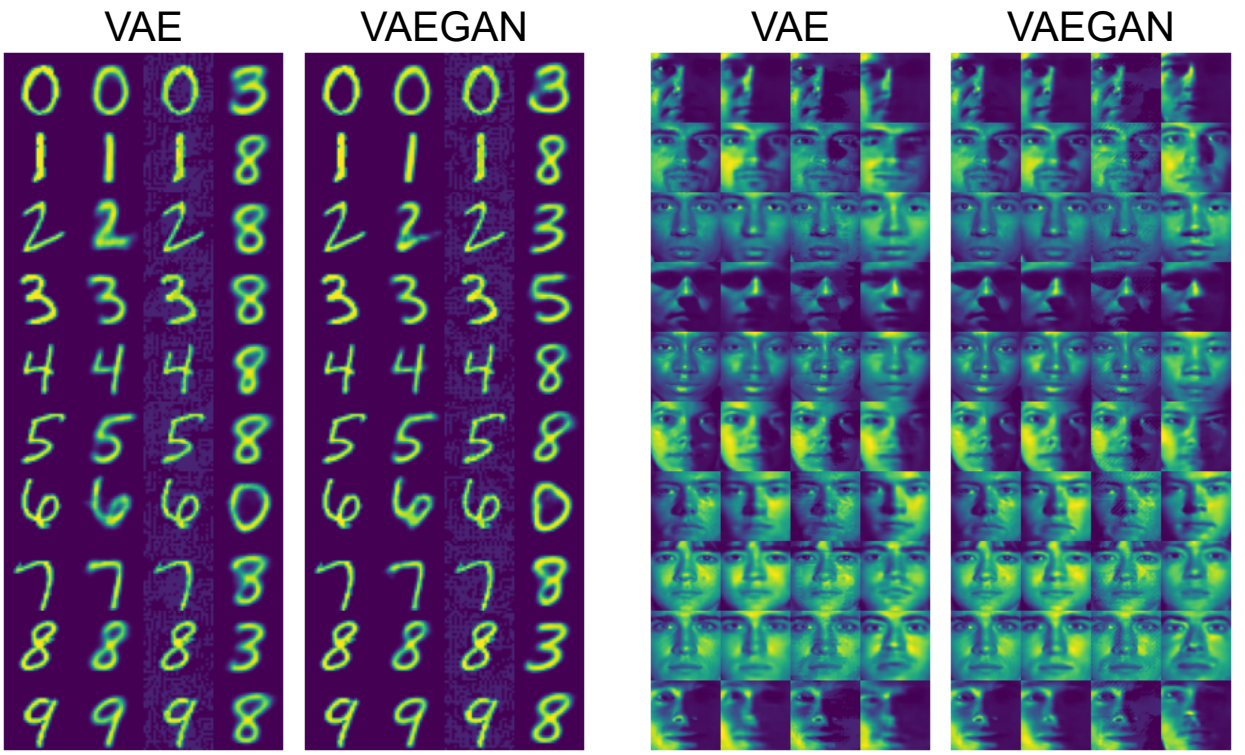
Fig. 3a and 3b show the results after evaluating the VAE and VAEGAN on the test samples of MNIST and CYFD with the standard framework. Note that the first two columns show the clean inputs and their reconstructed images of the models. The last two columns show the adversarial examples and their reconstructed images of the models. Noticeably, our generator can find weaknesses in both the VAE and VAEGAN.

4.3.2. Targeted Attack

When we trained our generator, we excluded training samples that belong to the targeted class because it did not update any parameter for the targeted-class samples. For MNIST, we picked all the classes to be the targeted classes. Therefore, we had ten generators, each of which is for each targeted class. For CYFD, we randomly picked only three classes (i.e., 2, 10 and 28) to be the targeted classes. Thus, we had three generators.

We evaluated our generators and a reconstruction model (i.e., VAE) on the test samples as shown in Table 2. Note that we excluded samples belonging to the targeted classes from the test samples to evaluate our generators fairly. Moreover, we consider that our generator successfully creates an adversarial example when its reconstructed image belongs to the targeted class.

According to Table 2, the classifier could achieve high accuracy on the reconstructed images of the clean samples on VAE with significantly low average confidences on the targeted classes. Further, FGSM achieved a very low success rate and average targeted confidence on every class. Then, PGD could achieve a slightly higher success rate and average targeted confidence. Essentially, our attack (SF) significantly outperformed those two baselines with a



(a) MNIST (b) CYFD

Fig. 3. The result of our standard framework on the left is from the VAE and on the right is from the VAEGAN. The images 1) in the first column are clean; 2) in the second column are the reconstructed images for the images in the first column; 3) in the third column are adversarial examples concerning the images in the first column; 4) in the last column are the reconstructed images for the adversarial examples.

Table 2. Results achieved by the classifier, the two baselines and our targeted attack performed by our standard framework on VAE in MNIST with each targeted class. Note that A R_X is the accuracy on R_X, TC R_X is the confidence on the targeted class on R_X, SR is the success rate and TC R_A is the confidence on the targeted class on R_A.

Class	A R_X	TC R_X	FGSM		PGD		SF	
			SR	TC R_A	SR	TC R_A	SR	TC R_A
0	91.04%	0.010	9.73%	0.10	28.02%	0.26	**73.46%**	0.72
1	90.63%	0.004	2.17%	0.02	11.31%	0.12	**28.76%**	0.29
2	91.73%	0.006	20.8%	0.21	41.29%	0.39	**81.79%**	0.78
3	91.94%	0.009	9.15%	0.09	22.45%	0.22	**75.16%**	0.74
4	91.78%	0.008	7.00%	0.07	23.26%	0.23	**47.75%**	0.47
5	92.59%	0.010	3.78%	0.40	11.58%	0.12	**48.54%**	0.47
6	91.17%	0.007	2.52%	0.30	13.70%	0.14	**36.12%**	0.35
7	91.86%	0.011	26.00%	0.26	42.77%	0.40	**76.57%**	0.74
8	92.40%	0.015	21.16%	0.21	44.01%	0.42	**83.55%**	0.82
9	91.95%	0.016	8.13%	0.08	31.42%	0.30	**75.90%**	0.74

much higher success rate and average targeted confidence on every class. Nonetheless, it achieved low success rates and average targeted confidence on some targeted classes (i.e., class 1, 4, 5 and 6) (still higher than the baselines). Implicitly, those targeted classes are distinct from the other classes. We also found similar results on VAEGAN.

Fig. 4 demonstrates the results from our generators attacking the VAE and VAEGAN. Note that the images on the left are adversarial examples generated by our generators from test samples, and the images on the right are their corresponding reconstructed images. Further, the images in column i belong to class i, and row i is the targeted class

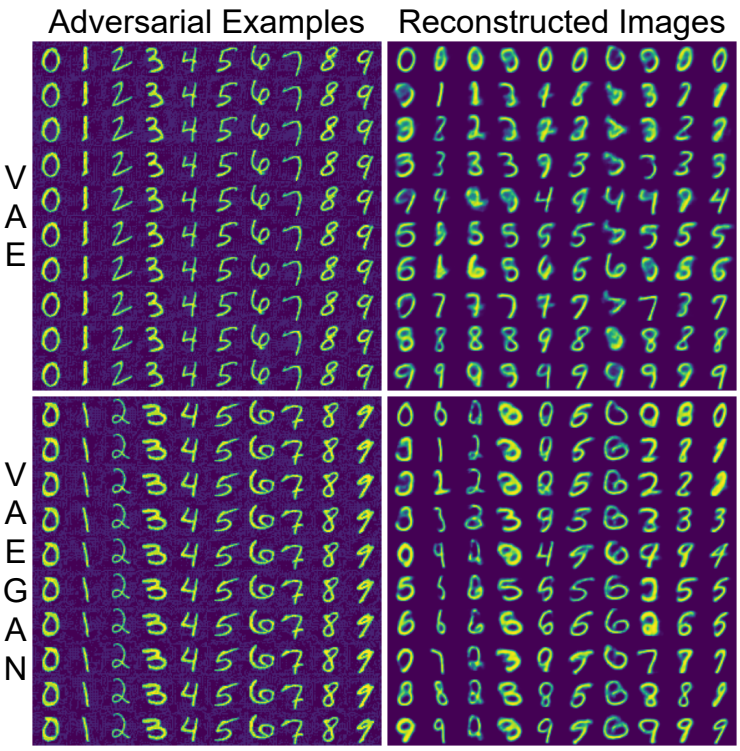


Fig. 4. Adversarial examples (left) and their corresponding reconstructed images (right) of the VAE (top) and VAEGAN (bottom). The columns represent images from class 0-9 respectively. The rows represent the targeted class 0-9 respectively.

i. Noticeably, our generators can successfully find perturbation to change the class of the reconstructed image to the desired class. Some classes are difficult to alter to other classes (e.g., class 0 and 6).

Table 3. Results achieved by the classifier, the two baselines and our targeted attack performed by our standard framework on VAE in CYFD with each targeted class. Note that A R_X is the accuracy on R_X, TC R_X is the confidence on the targeted class on R_X, SR is the success rate and TC R_A is the confidence (the final output after the softmax activation function) on the targeted class on R_A.

Class	A R_X	TC R_X	FGSM		PGD		SF	
			SR	TC R_A	SR	TC R_A	SR	TC R_A
2	76.31%	0.007	13.62%	0.12	36.69%	0.32	**41.51%**	0.38
10	75.83%	0.010	7.92%	0.07	23.96%	0.21	**33.33%**	0.31
28	76.53%	0.004	6.13%	0.06	20.72%	0.20	**21.78%**	0.2

However, in the entire test samples of CYFD, the generators on those targeted classes cannot achieve high success rates, as seen in Table 3, especially the targeted class 28. The reason is that humans' faces are very similar to each other. Thus, it is tough for the targeted attack to transform reconstructed images into its target. Nevertheless, our attack could outperform the baselines. We found similar results when the target was VAEGAN. Despite low success rates (still higher than the baselines), they can increase the classifier's confidence in the targeted classes by some significant amounts. Therefore, our generators are still effective on CYFD.

Further, Fig. 5 shows the results from our generators on the VAE and VAEGAN in CYFD. Note that the images on the left are the same in the top and bottom since they are from the targeted classes (i.e., 2, 10 and 28 respectively). The images in the middle are adversarial examples created by our generators and originally from class 0, 3, 6 and 14, respectively. Furthermore, the images on the right are their corresponding reconstructed images. Explicitly, our generator on the targeted class 2 can make the reconstructed images of class-0 and class-14 images similar to an image from class 2. Also, the targeted class 10's generator can make the reconstructed images' eyes of class-0, class-3 and

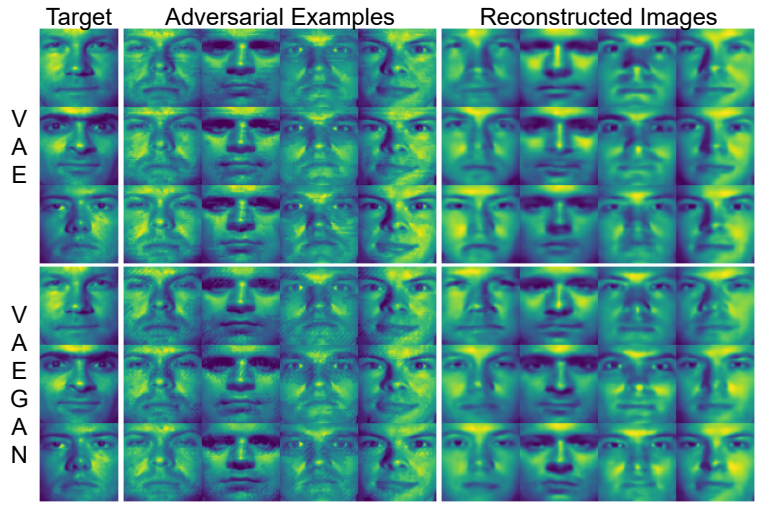


Fig. 5. Targeted-class images (left), adversarial examples (middle) and their corresponding reconstructed images (right) of the VAE (top) and VAEGAN (bottom) in CYFD

class-6 images similar to those from class 10. The generator on the targeted class 28 is also effective on images from class 0, 3 and 6.

4.4. Universal-Attack Framework

This section explores the performance of the universal-attack framework. We used the same setting as the standard framework during the training steps. The input size of the generator is 100 in our experiment. Note that the input of the generator in this framework is noise z.

Table 4. Accuracy achieved by the classifier on the sets of samples concerning MNIST and CYFD. Note that adversarial examples were generated by our universal-attack framework.

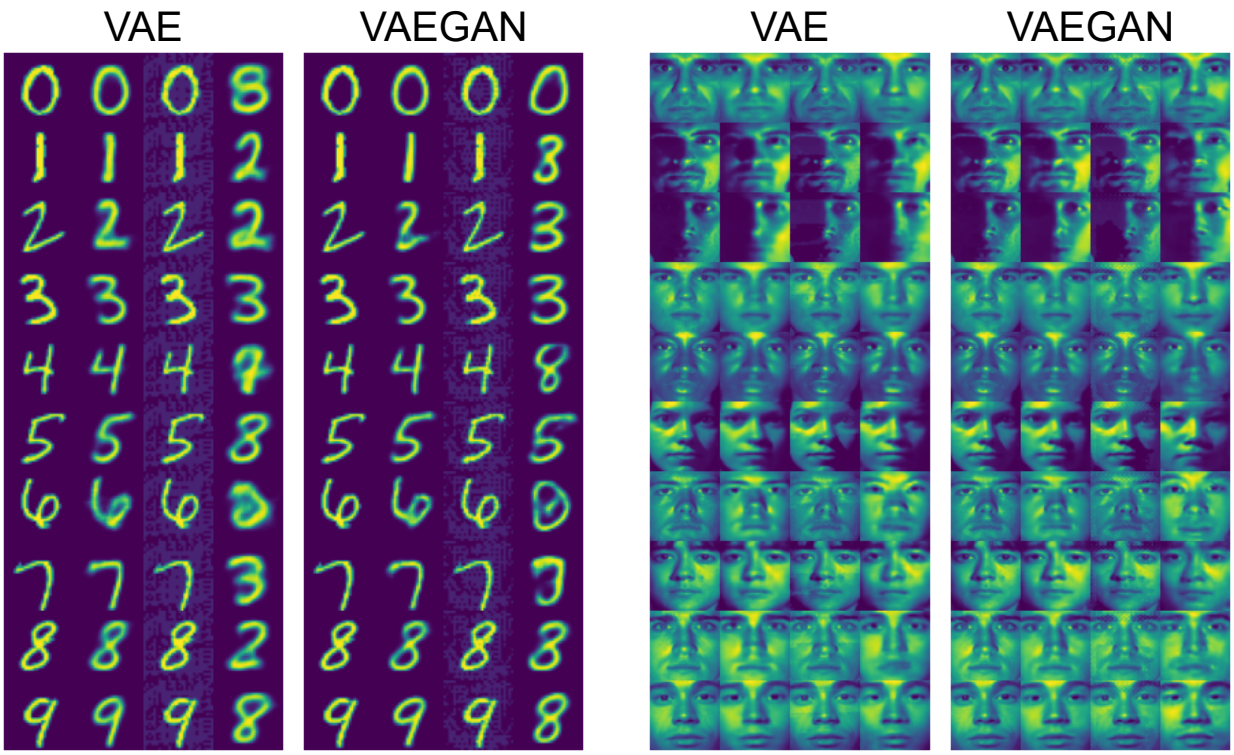
Model	MNIST				CYFD			
	X	R_X	A	R_A	X	R_X	A	R_A
VAE	99.47%	91.96%	99.28%	25.54%	99.39%	76.58%	73.93%	35.23%
VAEGAN	99.47%	91.89%	99.28%	25.46%	99.39%	84.73%	74.54%	24.24%

Although Table 4 shows that this universal-attack framework is highly effective in MNIST and CYFD, it is weaker than the standard framework when we compare their results in Table 1. The generator trained by the standard framework can reduce the accuracy more than the one trained by this framework. In addition, this universal-attack framework could outperform FGSM and PGD in MNIST even though these baselines required clean images as their inputs, as seen in Table 4 and 1. Also, in CYFD, it outperformed PGD. However, FGSM slightly outperformed this framework as seen in Table 4 and the column FGSM in Table 1.

Fig. 6 shows that the perturbation generated from this framework is less effective than the standard one when we compare it to Fig. 3. Furthermore, we also found that this framework was less effective than the standard one in the targeted attack. However, the universal-attack generator can perform better than the standard generator in real-time because it can prepare perturbation beforehand, regardless of the input.

4.5. Ablation Study

Since our framework consists of several components, we show what happens when we exclude the classifier and/or the discriminator. Then, we used the standard framework to find adversarial examples of VAE on MNIST to demonstrate the effect of lacking each part. Hence, we evaluated four frameworks: the complete standard framework (F_1),



(a) MNIST (b) CYFD

Fig. 6. The result of our universal-attack framework on the left is from the VAE and on the right is from the VAEGAN. The images 1) in the first column are clean; 2) in the second column are the reconstructed images for the images in the first column; 3) in the third column are adversarial examples concerning the images in the first column; 4) in the last column are the reconstructed images for the adversarial examples.

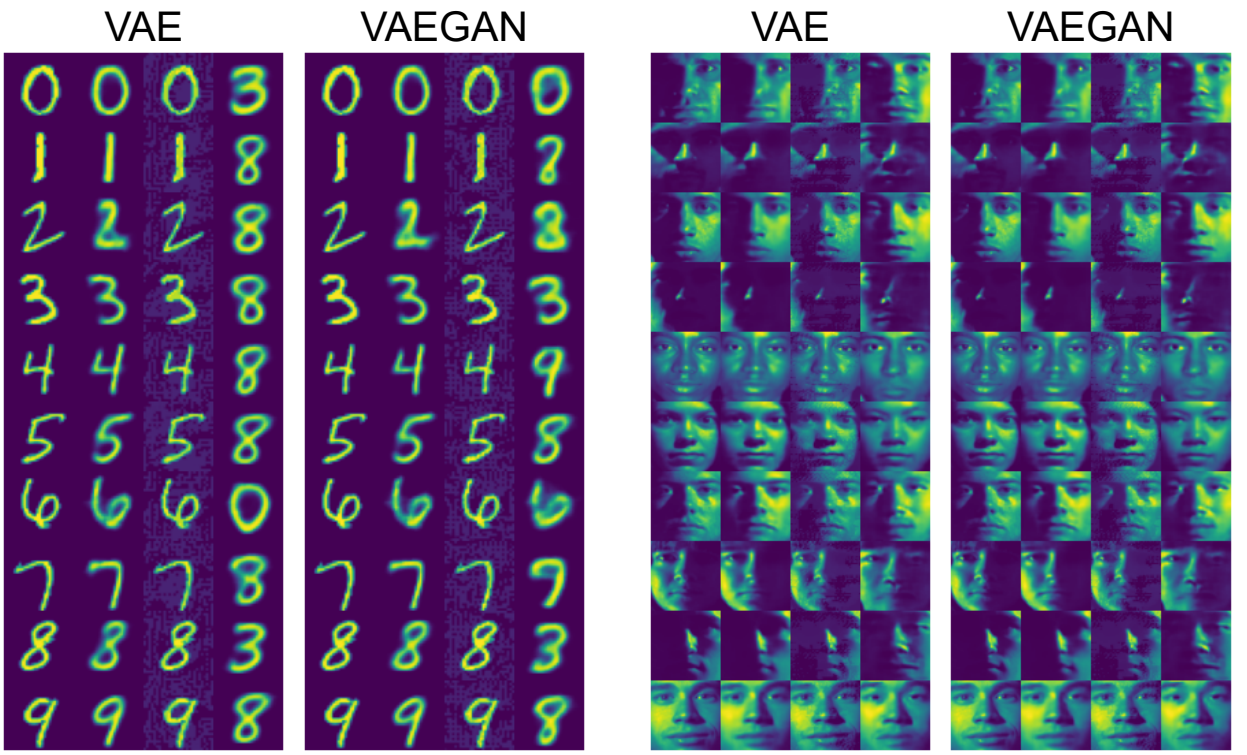
the standard framework without the discriminator (F_2), the standard framework without the classifier (F_3) and the standard framework without the discriminator and the classifier (F_4).

Table 5. Accuracy achieved by the classifier on the sets of samples from MNIST with VAE as the target. Note that adversarial examples were generated by our standard framework.

Framework	X	R_X	A	R_A
F_1	99.47%	91.51%	99.22%	4.84%
F_2			99.23%	4.04%
F_3			99.21%	76.65%
F_4			99.19%	18.72%

Table 5 shows the results after we trained and evaluated the frameworks (i.e., F_1, F_2, F_3 and F_4). Implicitly, by training the framework with the classifier, the generator could generate more effective adversarial examples than training it without the classifier due to the higher accuracy of the classifier on R_A. Further, training it with the discriminator slightly reduced the accuracy of the classifier on R_A. However, we found that training with the discriminator results in more valid reconstructed images than training without it.

Fig. 7 shows the results from the standard framework without the discriminator. Noticeably, even though the adversarial examples look valid, the adversarial examples from the standard framework with all the components in Fig. 3 are more valid than the ones from this framework without the discriminator.



(a) MNIST (b) CYFD

Fig. 7. The result of our standard framework without the discriminator on the left is from the VAE and on the right is from the VAEGAN. The images 1) in the first column are clean; 2) in the second column are the reconstructed images for the images in the first column; 3) in the third column are adversarial examples concerning the images in the first column; 4) in the last column are the reconstructed images for the adversarial examples.

5. Retraining for Robustness

In addition to finding adversarial examples, our framework can improve reconstruction models' robustness by utilizing the trained framework from the previous section. Intuitively, suppose a reconstruction model is robust against the untargeted attack. In that case, it will also be robust against the targeted attack since the untargeted attack will likely result in reconstructed images belonging to the easiest classes for their ground-truth classes. Similarly, suppose a reconstruction model is robust against the standard framework. In that case, it will be robust against the universal-attack framework because it is much more difficult to find adversarial examples than the standard one. Therefore, we used only the trained standard framework for the untargeted attack to retrain a pretrained reconstruction model to promote its robustness.

Note that we used only a pretrained VAE as our target for our experiment. For each epoch of our retraining, we first trained our framework for two epochs with the training mentioned in Section 3 and then trained the target for one epoch with the general training of VAE. However, the training samples of the target included the general training samples used in the previous section and the samples generated by our generator in the framework after feeding the general training samples. Hence, the target's training samples were twice more extensive than the general training samples.

To evaluate it, we used our untrained framework and trained it with the retrained target by following the instruction of the previous section. As a result, when we fed the samples created by our generators in the standard frameworks to the retrained target, we obtained the reconstructed images similar to the input. We achieved an accuracy of 89.5% in MNIST and 81.87% in CYFD on the reconstructed images produced by the retrained target in the standard framework. Furthermore, the classifiers also achieved high accuracy on the reconstructed images produced by the retrained target

Table 6. Accuracy achieved by the classifier on the sets of samples with retrained VAE as the target, attacked by the baselines and the universal-attack framework (UF)

Dataset	X	R_X	FGSM		PGD		UF	
			A	R_A	A	R_A	A	R_A
MNIST	99.47%	91.7%	99.05%	75.15%	99.41%	79.55%	99.12%	89.68%
CYFD	99.39%	89.21%	82.89%	48.07%	99.39%	49.49%	71.08%	70.67%

Table 7. Euclidean distance and cosine similarity between the latent spaces of VAE with standard dataset and the adversarial examples on MNIST and CYFD

Metric	No Retrain				Retrain			
	FGSM	PGD	SF	UF	FGSM	PGD	SF	UF
MNIST								
Euclidean	4972	4695	5855	5443	4689	4678	4618	4590
Cosine	0.72	0.75	0.67	0.71	0.76	0.76	0.76	0.76
CYFD								
Euclidean	267.58	236.95	267.53	262.35	283.86	264.73	256.57	270.89
Cosine	0.64	0.71	0.64	0.65	0.58	0.69	0.68	0.60

in both the universal framework and the baselines (i.e., FGSM and PGD), as demonstrated in Table 6. However, in CYFD, the robustness of the target against those baselines did not significantly improve in those baselines. They were still more robust than the original target. Moreover, we did not find any drawback in this retraining process because the accuracy of the classifier on R_X reconstructed by the retrained target was about the same as the one reconstructed from the natural target, as seen in Table 6.

Additionally, we also trained the standard frameworks with $\epsilon = 0.05$ and $\epsilon = 0.15$. Note that we used $\epsilon = 0.1$ for all the previous experiments on MNIST. We found that the framework with $\epsilon = 0.05$ did not work on the retrained VAE and the framework with $\epsilon = 0.15$ was less effective. However, the attack with $\epsilon = 0.15$ was too obvious.

6. Latent Space Analysis

This section experiments the latent spaces of a targeted model (i.e., VAE) on MNIST and CYFD. Table 7 shows the average differences between the latent spaces of VAE after passing the clean images and adversarial examples. Noticeably, the euclidean distance and cosine similarity follow the same pattern. That is, when the euclidean distance increases, the cosine similarity decreases. According to Table 1 and Table 7, even though SF can harm VAE the most, it also changes the latent spaces the most. Therefore, if an attacker needs the latent space to remain the same, we can add another factor to the loss function for it. Further, when we retrain VAE with our SF, as seen in Table 7, all the similarities increase.

For CYFD, in Table 7, the results of no-retrained VAE follows the same trend as in MNIST. Nonetheless, The results of the retrained VAE are interesting. Although the retrained VAE is more robust against those attacks than the no-retrained one, the cosine similarity achieved by FGSM, PGD and UF decrease from the no-retrained one. This phenomenon implies that the VAE for CYFD is not very organized since the latent vectors that are far away from the original ones can result in the similar reconstructed images.

7. Discussion and Conclusion

We proposed two GAN-based frameworks. We focused on attacking reconstruction models that reconstructed images that looked like their corresponding inputs and mathematically defined the problem. We discussed their corresponding loss functions and how to train standard and universal-attack frameworks. After training a generator in the standard framework, the framework needs an image input to find how to perturb the image within the specified perturbation bound to generate an adversarial example for a targeted reconstruction model. As a result, it is effective over MNIST on both untargeted and targeted attacks. Similarly, the strategy is also effective on CYFD for both attacks,

although not as good as in MNIST. Furthermore, we created two baselines based on state-of-the-art adversarial attacks in the literature (e.g., FGSM and PGD) for reconstruction models. Essentially, the proposed attack could outperform those baselines.

Moreover, we further experimented with the universal-attack framework. It trained a generator which is also effective in MNIST. Although it could not reduce the classification accuracy of the classifier as much as the one trained by the standard framework, it is perfect for attacking a target in real-time. Furthermore, from the ethical point of view, although our framework can be negatively used, we could successfully apply our framework to retrain reconstruction models to improve their robustness and empirically show that our attacks can no longer harm the retrained reconstruction models. Also, the retrained model could be robust against other attacks (i.e., FGSM and PGD).

Despite the success of our attack, it needs so much preprocessing time to train the whole framework. Hence, this preprocessing can be a significant limitation of this attack while FGSM and PGD do not require any training.

In addition to the reconstruction models, these frameworks can be applied to evaluate other applications that are image-to-image (e.g., interpretable keypoints from videos [30], anomaly detection [31], and graph prediction [32] by slightly adjusting the loss functions and their architectures. We leave this as future work.

References

[1] Q. Rao, J. Frtunikj, Deep learning for self-driving cars: Chances and challenges, in: Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems, 2018, pp. 35–38.

[2] I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, arXiv preprint arXiv:1412.6572.

[3] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, arXiv preprint arXiv:1312.6199.

[4] N. Papernot, P. McDaniel, X. Wu, S. Jha, 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] W. Xu, D. Evans, Y. Qi, Feature squeezing: Detecting adversarial examples in deep neural networks, arXiv preprint arXiv:1704.01155.

[6] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, arXiv preprint arXiv:1706.06083.

[7] S.-M. Moosavi-Dezfooli, A. Fawzi, P. Frossard, Deepfool: a simple and accurate method to fool deep neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2574–2582.

[8] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, A. Swami, The limitations of deep learning in adversarial settings, in: 2016 IEEE European symposium on security and privacy (EuroS&P), IEEE, 2016, pp. 372–387.

[9] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, A. Swami, Practical black-box attacks against machine learning, in: Proceedings of the 2017 ACM on Asia conference on computer and communications security, 2017, pp. 506–519.

[10] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: 2017 ieee symposium on security and privacy (sp), IEEE, 2017, pp. 39–57.

[11] C. Lassner, G. Pons-Moll, P. V. Gehler, A generative model of people in clothing, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 853–862.

[12] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, I. Sutskever, Jukebox: A generative model for music, arXiv preprint arXiv:2005.00341.

[13] B. Samanta, A. De, G. Jana, V. Gómez, P. K. Chattaraj, N. Ganguly, M. Gomez-Rodriguez, Nevae: A deep generative model for molecular graphs, Journal of machine learning research. 2020 Apr; 21 (114): 1-33.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Communications of the ACM 63 (11) (2020) 139–144.

[15] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, D. Song, Generating adversarial examples with adversarial networks, arXiv preprint arXiv:1801.02610.

[16] T. Bai, J. Zhao, J. Zhu, S. Han, J. Chen, B. Li, A. Kot, Ai-gan: Attack-inspired generation of adversarial examples, in: 2021 IEEE International Conference on Image Processing (ICIP), IEEE, 2021, pp. 2543–2547.

[17] P. Tabacof, J. Tavares, E. Valle, Adversarial images for variational autoencoders, arXiv preprint arXiv:1612.00155.

[18] J. Kos, I. Fischer, D. Song, Adversarial examples for generative models, in: 2018 ieee security and privacy workshops (spw), IEEE, 2018, pp. 36–42.

[19] P. Pope, Y. Balaji, S. Feizi, Adversarial robustness of flow-based generative models, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 3795–3805.

[20] L. Deng, The mnist database of handwritten digit images for machine learning research, IEEE Signal Processing Magazine 29 (6) (2012) 141–142.

[21] K. Lee, J. Ho, D. Kriegman, Acquiring linear subspaces for face recognition under variable lighting, IEEE Trans. Pattern Anal. Mach. Intelligence 27 (5) (2005) 684–698.

[22] D. P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114.

[23] X. Yu, X. Zhang, Y. Cao, M. Xia, Vaegan: A collaborative filtering framework based on adversarial variational autoencoders., in: IJCAI, 2019, pp. 4206–4212.

[24] F. Chollet, Variational autoencoder, https://keras.io/examples/generative/vae/, accessed: 2020-05-03.
[25] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, arXiv preprint arXiv:1511.06434.
[26] A. Jang, Tensorflow: Mnist cnn tutorial, https://www.kaggle.com/amyjang/tensorflow-mnist-cnn-tutorial, accessed: 2020.
[27] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.
[28] M. Stewart, Gans vs. autoencoders: Comparison of deep generative models, https://towardsdatascience.com/gans-vs-autoencoders-comparison-of-deep-generative-models-985cf15936ea, accessed: 2019-05-12.
[29] Z. Liu, P. Luo, X. Wang, X. Tang, Deep learning face attributes in the wild, in: Proceedings of International Conference on Computer Vision (ICCV), 2015.
[30] T. Jakab, A. Gupta, H. Bilen, A. Vedaldi, Self-supervised learning of interpretable keypoints from unlabelled videos, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 8787–8797.
[31] C. Zhou, R. C. Paffenroth, Anomaly detection with robust deep autoencoders, in: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, 2017, pp. 665–674.
[32] P. V. Tran, Learning to make predictions on graphs with autoencoders, in: 2018 IEEE 5th international conference on data science and advanced analytics (DSAA), IEEE, 2018, pp. 237–245.