AdvGAN vs. Defensive Purification

Fan Yang **Nao Rho** **Junran Yang**

1 Introduction and Motivation

Deep learning is providing major breakthroughs in solving many problems that have withstood attempts from the machine learning and artificial intelligence community in the past. Despite their high accuracy, deep neural networks have been shown to be vulnerable to *adversarial attacks* in the context of image classification. Such attacks add small perturbations that cannot be perceived by human eyes to pristine images and cause neural network classifiers to completely change its prediction with high confidence. For instance, a hacker can construct an image of a $100 check that looks harmless to a banker or a machine, but, with careful construction, a machine learning algorithm recognizes it as $999 with high confidence. Such attack is detrimental to industrial applications and must be dealt beforehand.

Two types of threats are considered by most of the recent literature, *white-box* and *black-box* attacks. Under the white-box attack, the attacker has full access to the architecture and parameters of the classification model and the defense mechanisms; for the black-box attacks, the attackers do not know any of the information. Although, there is not a universally accepted solution for defense strategy, of all the methods proposed, they can be grouped into three approaches. For our project, we focused on one of them that we'd like to call *defensive purification* because of its superiority demonstrated by its effectiveness against both white-box and black-box attacks, its high transferability against new attacks that it is not designed for, and also its easy installation as a pre-processing unit for different types of classifiers without modifying any parameters of the trained classifiers. An overview of the defense mechanism using a purifier can be seen in Figure 1. The adversarial example goes through a purifier that diminish its perturbation before they are fed into the classifier.

The two purifiers with different inner structure we used for this approach are *PuVEA* and *defense-GAN*. We tested them against the attack model *advGAN*, which we are going to discuss in later sections. The rest of the report is organized as follows. We introduce the backgrounds of the models in Section 2. Our experiment method against different defense model will be given in Section 3. Finally, results analysis as well as the comparisons will be discussed in Section 4.

2 Background

2.1 PuVAE

Purifying Variational Autoencoder(PuVAE) is a conditional VAE that is designed to purify the inputs. The general idea is that PuVAE folds encodes inputs to latent variables, which are then translated back to the purified inputs. During the encoding step, PuVAE is effectively throwing out adversarial noises that are unnecessary. In detail, PuVAE utilizes a condtional VAE(cVAE) as its encoding and decoding. The encoder of PuVAE takes the input of both the image and the label and outputs latent codes, which are sampled from the gaussian distribution with the μ and σ defined by the last layer of the encoder. These latent codes and the corresponding label are fed to the decoder. During the training step, the decoded output image is then passed into a pre-trained source classifier. During the purified step, the decoded output is passed into the target classifier instead. During the training step, one might wonder the necessity of source classifier. The source classifier is placed in PuVAE to ensure that the conditional VAE reconstruct images that are correct to the labels.

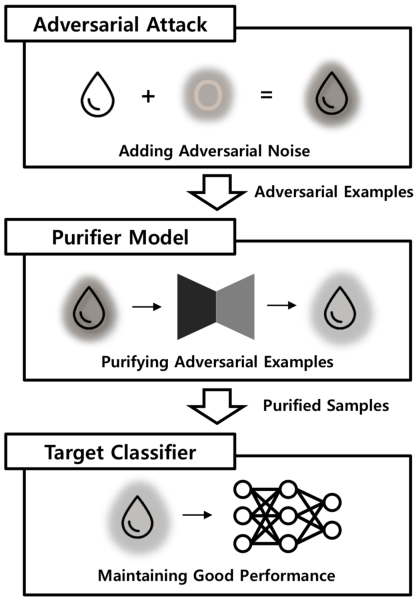


Figure 1: Overview of the defense mechanism using the purifier model.

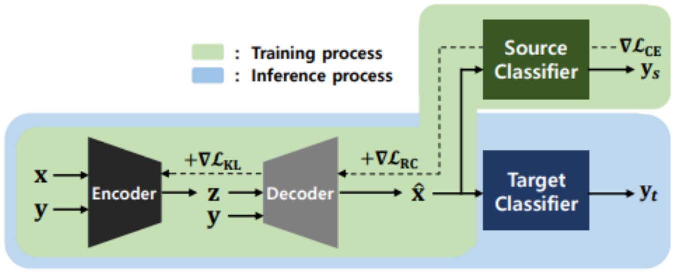


Figure 2: information flow of PuVAE

The loss function is defined quite interestingly. PuVAE loss is made of three losses. They are listed below.

$$\mathcal{L}_{RC} = x \log \hat{x} + (1 - x) \log(1 - \hat{x}) \tag{1}$$

$$\mathcal{L}_{KL} = \mu^2 + \sigma^2 - \log(\sigma^2 - 1) \tag{2}$$

$$\mathcal{L}_{CE} = y \log y_s + (1 - y) \log(1 - y_s) \tag{3}$$

$\mathcal{L}_{RC}$ is the reconstruction loss which is binary cross entropy loss between the real image(x) and the reconstructed image($\hat{x}$). This has the effect of making x and $\hat{x}$ close to each other. $\mathcal{L}_{KL}$ is the KLD loss that appears often in VAE models. There are a lot of ways to explain this loss, but it makes the latent variable meaningful when reconstructing $\hat{x}$. The last loss, $\mathcal{L}_{CE}$, is the binary cross entropy loss between the input label and the output label from the source classifier. This loss has the effect of making sure that the output image of decoder belongs to the certain label. The final PuVAE loss is defined as:

$$\mathcal{L}_{PuVAE} = \mathcal{L}_{RC}\lambda_{RC} + \mathcal{L}_{KL}\lambda_{KL} + \mathcal{L}_{CE}\lambda_{CE} \tag{4}$$

Here, $\lambda_{KL}, \lambda_{RC}$, and λ_{CE} are the constant for each loss. Controlling these parameters have the effect of making one loss more significant than the others.

2.2 DefenseGAN

DefenseGAN uses a GAN trained on real unperturbed samples as the purifier. During the training time, GAN is designed to model the distribution of the real samples. It's expected that real samples will be close to some point in the range of $\mathcal{G}$, whereas the adversarial examples will be further away. Prior to feeding any input to the classifier, the generator project the input to its domain, ie., the latent space, to diminish the effect of perturbations. The projected output is then fed into the classifier instead of the perturbed image.

At inference time, with the generator G of the trained GAN, we need to find $\mathbf{z}^*$ that minimize the reconstruction error $\|G(\mathbf{z}) - \mathbf{x}\|_2^2$, where $\mathbf{x}$ is the image to be classified. $G(\mathbf{z}^*)$ is then fed into the the classifier as an input. Since the minimization of the reconstruction error is a highly non-convex optimization problem, we approximate it by L iterations of Gradient Descent with R different randomly generated noise z. The procedure is illustrated in Figure 3.

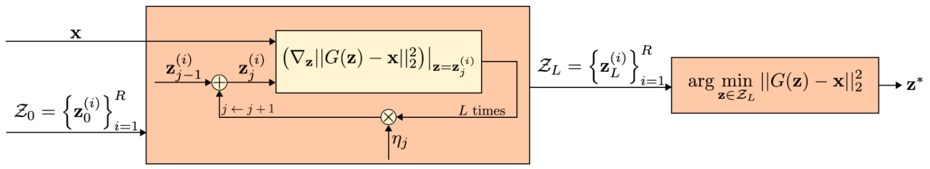

$\mathbf{x}$
$\mathcal{Z}_0 = \left\{\mathbf{z}_0^{(i)}\right\}_{i=1}^R$
$\mathbf{z}_{j-1}^{(i)}$
$\mathbf{z}_j^{(i)}$
$\left(\nabla_{\mathbf{z}}\|G(\mathbf{z}) - \mathbf{x}\|_2^2\right)\big|_{\mathbf{z}=\mathbf{z}_j^{(i)}}$
$j \leftarrow j+1$
L times
η_j
$\mathcal{Z}_L = \left\{\mathbf{z}_L^{(i)}\right\}_{i=1}^R$
$\arg\min_{\mathbf{z}\in\mathcal{Z}_L} \|G(\mathbf{z}) - \mathbf{x}\|_2^2$
$\mathbf{z}^*$

Figure 3: L steps of Gradient Descent are used to project the image to the range of the generator G

Under this setting, DefenseGAN can be used in conjunction with any classifier without modifying its structure. Because it does not assume any attack models but built on the idea of GAN generator, it can be used to any attack. One thing to notice is that one might come across practical difficulties using DefenseGAN because training GANs still remains to be a challenging problem. Moreover, the fine-tuning of hyper-parameters L and R is also critical to the performance of this defense.

2.3 AdvGAN

AdvGAN is a model published by a group of researchers at Michigan and Berkeley in 2018. It's highly cited and recognized as one of the best attack techniques today. Figure 3 illustrates the overall architecture of AdvGAN. The network mainly consists of three parts: a generator $\mathcal{G}$, a discriminator $\mathcal{D}$, and a target (under white-box setting) or a distilled (under black-box setting) classifier f. The generator $\mathcal{G}$ takes in the original image x and generates a perturbation $\mathcal{G}(x)$. The perturbed image $x + \mathcal{G}(x)$ is sent to the discriminator $\mathcal{D}$ whose goal is to encourage that the perturbed image is indistinguishable from the original image. The perturbed image is also sent to the classifier f to output a loss $\mathcal{L}_{adv}$, which represents the distance between the prediction and the target class t (targeted attack), or the opposite of the distance between the prediction and the ground truth class (untargeted attack).

The adversarial loss $\mathcal{L}_{GAN}$ can be written as

$$\mathcal{L}_{GAN} = \mathbb{E}_x \log \mathcal{D}(x) + \mathbb{E}_x \log(1 - \mathcal{D}(x + \mathcal{G}(x))) \tag{5}$$

Here, the discriminator $\mathcal{D}$ aims to distinguish the perturbed data $x + \mathcal{G}(x)$ from the original data x. The loss for fooling the classifier f is

$$\mathcal{L}_{adv}^f = \mathbb{E}_x \ell_f(x + \mathcal{G}(x), t) \tag{6}$$

For targeted attacks, t is the target class and ℓ_f is the loss function used to train the model f. For untargeted attacks, t is the ground truth label and ℓ_f is opposite of the loss used to train f. To bound the perturbation $\mathcal{G}(x)$ within a range, a soft hinge loss of the L_2 norm is used:

$$\mathcal{L}_{hinge} = \mathbb{E}_x \max(0, \|\mathcal{G}(x)\|_2 - c) \tag{7}$$

where c denotes a user-specified bound. The full objective of AdvGAN can be expressed as

$$\mathcal{L} = \mathcal{L}_{adv}^f + \alpha\mathcal{L}_{GAN} + \beta\mathcal{L}_{hinge}) \tag{8}$$

where α and β control the relative importance of each objective. Once $\mathcal{G}$ is trained on the training data using f, it can generate perturbations efficiently for any input.

AdvGAN has achieved high attack success rate under state-of-the-art defenses compared to other attacks. On Madry et al.'s MNIST challenge (2017), AdvGAN achieved 88.93% accuracy on the published robust model in the whitebox setting and 92.76% in the black-box setting, which was the top position in the challenge.

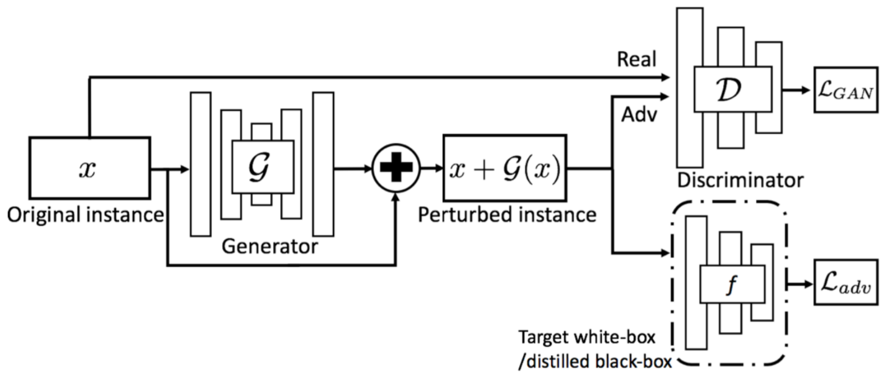


Figure 4: Overview of advGAN

3 Method

In this project, we used three datasets: MNIST, Fashion MNIST, and CIFAR10. We trained a classifier for each of the datasets, then tested the performance of the classifiers with and without defenses. Our experiments can be grouped into two groups. For the first group we assume attacker don't know the parameters of the defensive network, and for the second group we assume attacker have that information. For both groups of our experiments we assume attackers know the parameters of the target classifiers. Note that usually for black-box attacks attackers have information about neither the target classifiers nor the defensive networks. However, since there have been abundant work published in that area, we figured out it's more interesting to explore performance of the target classifiers while only parameters of the defensive network are unknown. This type of attacks may be called "semi-black-box attacks" in an accurate way, but we will call them "black-box attack" in the rest of the report for abbreviation. The only "full" black-box experiments we conducted the black-box attacks on the target classifiers with no defenses, as shown in the first column of table 2.

3.1 Architecture

In this section we briefly introduce internal structures of all the networks used in our experiments.

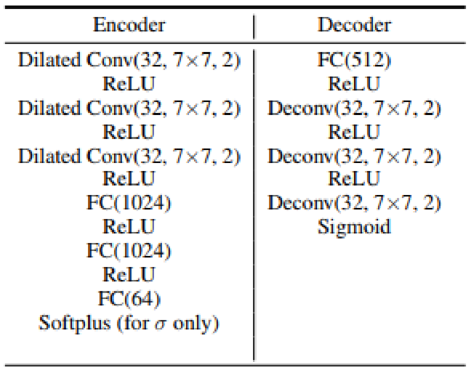
3.1.1 Target Classifiers

The target classifiers for MNIST and Fashion MNIST datasets share the same internal structure, which consists of two conv2d layers followed by a max pooling layer, then another two conv2d layers followed by max pooling, then a dense layer at the end. The target classifier for CIFAR10 dataset is the VGG13 network published in [8]. We used the VGG13 structure without modifications.

3.1.2 PuVAE

In this paper, we used the same parameters as the original paper. Below is the architecture for PuVAE.

32, 7×7, 2 of the encoder indicates the number of channel, filter size, and the dilation. Likewise, 32, 7×7, 2 of the encoder indicates the number of channel, filter size, and the stride. FC stands for "Fully Connected Layer". The size of latent codes were varied for different datasets. We used 50, 100, 256 for MNIST, FMNIST, and CIFAR10 each. We varied the size of latent codes to increase the purification potentials. The architecture for source classifier is described above.



Encoder	Decoder
Dilated Conv(32, 7×7, 2)	FC(512)
ReLU	ReLU
Dilated Conv(32, 7×7, 2)	Deconv(32, 7×7, 2)
ReLU	ReLU
Dilated Conv(32, 7×7, 2)	Deconv(32, 7×7, 2)
ReLU	ReLU
FC(1024)	Deconv(32, 7×7, 2)
ReLU	Sigmoid
FC(1024)	
ReLU	
FC(64)	
Softplus (for σ only)	

Figure 5: PuVAE architecture

3.1.3 DefenseGAN

The gradient descent step of defenseGAN was implemented with PyTorch. We applied 10 random restarts and 200 iterations of SGD in each random restart for optimization of the randomly initialized latent variables. For more details on the The generator of defensive GAN was implemented with DCGAN [7]. For the generator, we used the architecture from the original DCGAN paper[7], which consists of four layers of base units. Each base unit consists of a convolutional layer with a 4×4 filter, a batch normalization layer, and an ReLU layer. The channel sizes were set to be 8, 16, 32, 64, and 1.

3.1.4 AdvGAN

For the generator of AdvGAN we used 5 conv2d layers interleaved with 5 InstanceNorm2d layers to project the inputs to latent variables of length 32 (for MNIST and Fashion MNIST datasets) or 96 (for CIFAR10), then we use 5 upsampling layers interleaved with another 5 InstanceNorm2d layers to project the latent variables back to the input space. For the discriminator we used 3 or 4 conv2d layers interleaved with 2 InstanceNorm2d layers, followed by a dense layer that makes the prediction. Details of the layers are not described here due to space limits but can be provided upon request.

3.2 Experiments

We conducted a total of 7 experiments for each dataset, which are described below.

1. We performed the baseline accuracy measurements. We measured the accuracy of target classifier under no attack and no defense.
2. We performed black box attacks using advGAN on our target classifier with no defense, defensed by PuVAE, and defensed by defenseGAN. This is the case when the attacker does not know the presence of the defense methods. We pre-trained generators using advGAN and the target classifiers, then used the trained generators to generate adversarial images, which were then fed to the defensive networks. The purified images were taken in by the target classifiers to give the black-box attack results.
3. We performed the white box attack with advGAN on our target classifier with no defense, defensed by PuVAE, and defensed by DefenseGAN. This is the case when the attacker knows the architecture and the parameters of the defense methods. To get the highest possible attack success rates under this setting, we performed training of advGAN with the target classifiers and the defensive networks tied together. That is, we added the defensive network of interest right before the target classifier f in Figure 4, then performed a training. White-box attack results were output and updated by the target classifier f as the training went on.

All of the attacks we conducted were untargeted. We used a perturbation bound of 0.2, which means every pixel of the original images can be altered by at most 0.2 on a [0, 1] scale. This is a quite strict constraint and under this constraint, the perturbed images are usually not distinguishable from the original images in human eyes.

Since the models we used in this project were all published very recently (AdvGAN and PuVAE in 2019, DefenseGAN in 2018), there have never been reported combats or comparisons between them. Our results represent the earliest assessments of their performances when tested against each other.

4 Result & Discussion

We show the performance of the target classifiers under no attack and no defense, and the attack success rates on the target classifiers by AdvGAN under black-box and white-box settings, in tables 1-3, respectively.

MNIST	FashionMNIST	CIFAR10
99.5	92.4	88.6

Table 1: Accuracy of target classifiers under no attack

dataset	No defense	PuVAE	DefenseGAN
MNIST	56.6	**17.2**	34.0
FashionMNIST	65.2	**22.5**	54.8
CIFAR10	23.3	**16.0**	20.7

Table 2: Attack success rates on target classifiers under black-box setting

dataset	No defense	PuVAE	defenseGAN
MNIST	93.8	**74.1**	90.6
FashionMNIST	99.1	**92.2**	95.3
CIFAR10	96.7	91.1	**87.5**

Table 3: Attack success rates on target classifiers under white-box setting

As we can see, AdvGAN were successful at reducing defense accuracy by a large amount with the classification networks we used, especially for the white-box attacks. However, both defense mechanisms showed their ability of diminishing the attack success rate of AdvGAN. In terms of comparison between the two defensive networks, both of our black-box and white-box experiments showed that PuVAE performed significantly better than DefenseGAN. The results are surprising because DefenseGAN has been recognized as one of the best defense methods, while PuVAE is a newly proposed model that has not gained much attention. The under-satisfactory performance of DefenseGAN may be accounted by the fact that we utilized the default parameter L and R from the original DefenseGAN paper without further fine-tuning. However, taking into consideration the obstacle of optimizing parameters for DefenseGAN on every different dataset, PuVAE seems to be a more robust and appealing defense method. Speed-wise, DefenseGAN was approximately 100 times slower than PuVAE because for each input the generator of DefenseGAN has to go through $\mathbf{L} \times \mathbf{R}$ ($\mathbf{R} = 10$, $\mathbf{L} = 200$ in the published code) steps of Gradient Descent. In fact, it is unfair to compare PuVAE and Defense-GAN without time constraint; if we limit the prediction time to be a reasonably small number, e.g. 100 milliseconds, the advantage of PuVAE over DefenseGAN would become more significant. Therefore, PuVAE is more practical in real world scenarios for its much better performance in a reasonable time condition.

5 Conclusion

In this paper, we have tested the efficiency of the state-of-art defense method, DefenseGAN and PuVAE, against the state-of-art attack method, AdvGAN. We obtained surprising result that the unpopular PuVAE showed better performance than the well-known DefenseGAN. There are some limits of this project report as we predominantly used the default parameters from the published code and the paper due to limits in our time and our access to computational resources. We can also consider replacing the normal DCGAN with a stabler, easier-to-train WGAN for training the generator of DefenseGAN. That being said, this study has well demonstrated efficiency of some state-of-art defense/attack techniques. As for the future work, we hope to use different target classifiers at training time and testing time so we can examine the transferability of these two defensive networks.

References

[1] Uiwon Hwang, Jaewoo Park, Hyemi Jang, Sungroh Yoon: “PuVAE: A Variational Autoencoder to Purify Adversarial Examples”, 2019; [http://arxiv.org/abs/1903.00585 arXiv:1903.00585].

[2] 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.

[3] Pouya Samangouei, Maya Kabkab: “Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models”, 2018; [http://arxiv.org/abs/1805.06605 arXiv:1805.06605].

[4] George Gondim-Ribeiro, Pedro Tabacof: “Adversarial Attacks on Variational Autoencoders”, 2018; [http://arxiv.org/abs/1806.04646 arXiv:1806.04646].

[5] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu: “Generating Adversarial Examples with Adversarial Networks”, 2018; [http://arxiv.org/abs/1801.02610 arXiv:1801.02610].

[6] J. Kos, I. Fischer and D. Song, "Adversarial Examples for Generative Models," 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, 2018, pp. 36-42. doi: 10.1109/SPW.2018.00014

[7] Alec Radford, Luke Metz: “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, 2015; [http://arxiv.org/abs/1511.06434 arXiv:1511.06434].

[8] Karen Simonyan, Andrew Zisserman: "Very Deep Convolutional Networks for Large-Scale Image Recognition", 2014; [https://arxiv.org/abs/1409.1556 https arxiv:1409.1556]