Generating adversarial examples using Generative Adversarial Network (GAN)

Yeganeh Morshedzadeh

***Abstract*—Since much research has been made on the vulnerability of Deep neural networks (DNNs), we tried to generate more realistic and efficient adversarial examples in this project. More precisely, we used the Adv-GAN framework to learn and approximate the distribution of original instances. Afterward, we used the generator to produce high perceptual quality adversarial examples once the Adv-GAN was trained. The generator can be efficiently and independently used to generate adversarial perturbations for any given instance. Therefore, this attack can be used in both semi-white-box and black-box attacks. Using this method we could perform a 99.57% attack success rate on the target model, which had a 99.3% accuracy on the MNIST dataset before the attack.**

***Index Terms*—Generative Adversarial Network, Adversarial Example, Fast Gradient Sign Method, Adversarial Perturbations, Deep neural networks.**

I. Introduction

CUrrently, deep learning (DL) along with other machine learning (ML) and artificial intelligence (AI) methods have made remarkable advancements in providing solutions to challenging scientific problems at a large scale, such as reconstructing brain circuits [1] and analyzing mutations in DNA [2]. In addition, deep neural networks (DNNs) have become the preferred choice of researchers when tackling many challenging problems in speech and voice recognition [3], natural language understanding [4], and computer vision.

Continuous advancements in deep neural network models [5], [6], open access to DL software libraries [7]–[9], and easy access to the required hardware for training complex models have helped DL quickly reach a level of maturity for entry into sensitive and critical applications such as self-driving cars, surveillance systems [10], malware detection [11], [12], robots and drones [13], [14], voice command recognition [3], and face ID security on mobile devices.

In some cases, these applications are so important that an individual's life and property depend on them [15]. For example, an attacker could mislead an autonomous vehicle or compromise the intelligent agents that communicate via voice commands. Research has shown that machine learning algorithms are vulnerable to adversarial examples. These examples, obtained by slight and imperceptible perturbations in the dataset samples, can mislead the classifier model [16].

Considering that in most research, the goal has been to increase the accuracy of the network and less attention has been paid to the security and flexibility of the model, even for the most powerful trained networks with very high accuracy in image classification, one can find adversarial examples that mislead the network into predicting with confidence and misleading it. Therefore, ensuring the robustness and security of deep learning models is of great importance.

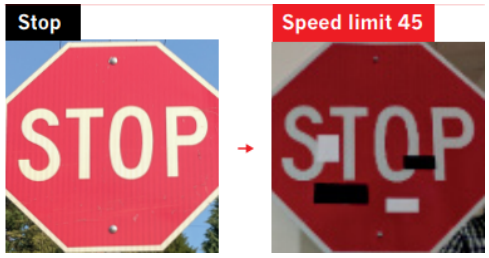


Fig. 1. An example of error in neural networks in detecting traffic signs [17].

An example of the importance of adversarial examples.

Imagine a self-driving car approaching an intersection and approaching a stop sign but instead of reducing speed, it increases its speed and as a result, an accident occurs. Later, a reporter reveals that there were four rectangles glued onto the stop sign, which caused the AI of the car to mistakenly recognize it as a speed limit of 45. This real example was presented by scientists who were able to mislead the AI system by placing labels on the stop sign [17]. In the image on the left of fig. 1, by placing labels, the image is transformed into the image on the right, which, although easily recognizable by humans, is recognized by AI systems as a 45 speed limit sign instead of a stop sign, which is very dangerous [18].

II. Background

Adversarial Example & Adversarial Training

An adversarial example/image is a modified version of a sample, for example, an image that is intentionally and knowingly distorted and manipulated to mislead the network.

These examples are created by adversarial perturbations, which are small changes in the original image that are imperceptible or quasi-imperceptible to humans but appear entirely different to networks.

Adversarial training is the use of adversarial examples alongside clean and untouched samples to train deep learning models.

Classification of Adversarial Attacks

Adversarial attacks are classified into different categories, such as targeted attacks and untargeted attacks, universal attacks and data-dependent attacks, perturbation attacks and replacement attacks, black-box attacks, semi-white-box attacks, and white-box attacks.

White-Box Attacks: In this type of attack, it is assumed that complete knowledge about the target model, parameter values, architecture, training method, and in some cases, training data, is available.

Black-Box Attacks: In black-box attacks, adversarial examples that are created without any knowledge of the target model are presented to the target model during the testing phase. In some cases, it is assumed that the attacker has limited knowledge about the model, such as the training process or architecture, but certainly knows nothing about the model's parameters.

Semi-White-Box Attacks: In this type of attack, which is very similar to black-box and white-box attacks, any other knowledge or information about the target model is used. For example, probabilities obtained from the network's predictions can be used as additional information whereas in black-box attacks could be using only the final prediction, i.e. predicted class.

Generative Adversarial Network

Generative Adversarial Network (GAN) [19], consists of two main parts called generator and discriminator. These two neural networks have opposite functions and compete with each other in a zero-sum game. Originally, this network was introduced as a generative model for unsupervised learning, but generative adversarial networks have proven to be useful for semi-supervised learning, supervised learning, and reinforcement learning.

Generator: The generator takes a noise, often in the form of Gaussian or uniform, as its input and then produces a very noisy image of the input data. The main goal of the generator is to produce images that are as similar as possible to natural and realistic images in the dataset.

The generator produces images with the same dimensions as the images in the dataset from random noise.

Discriminator: The discriminator takes an image as its input and its task is to distinguish between real images, images in the dataset, and fake images, generated by the generator.

If the discriminator recognizes the image as natural, it gives a value close to 0 in the output, and if it recognizes the image as unnatural, it gives a value close to 1 in the output.

How does a GAN work?: The main goal of the Generative Adversarial Network is to generate an image that is as natural as possible to the extent that it can deceive both computers and humans. As shown in fig. 2, the images generated by the generator and the images in the dataset are fed to the discriminator. The discriminator's goal is to distinguish between the two types of images, while the generator's goal is to produce images that can deceive the discriminator. Specifically, the generator first produces some natural and convincing images, and then after calculating the gradients, these values are used to update the parameters of both networks.

In fig 3, a visualization of the interaction and training process of a generative adversarial network is shown[1].

[1]This can be accessed through https://poloclub.github.io/ganlab/

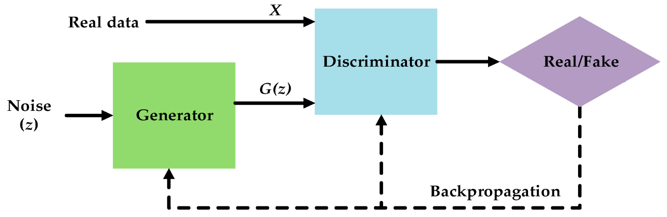
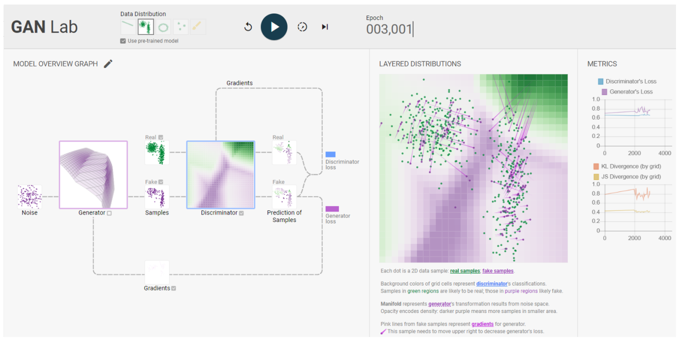


Fig. 2. Architecture of a GAN [19]



Each dot is a 2D data sample: real samples; fake samples.

Background colors of grid cells represent discriminator's classifications. Samples in green regions are likely to be real; those in purple regions likely fake.

Manifold represents generator's transformation results from noise space. Opacity encodes density: darker purple means more samples in smaller area.

Pink lines from fake samples represent gradients for generator.
This sample needs to move upper right to decrease generator's loss.

Fig. 3. Visualization of how a GAN works.

III. Review of Related Work

To attack neural networks, DNNs can be exposed to adversarial examples. By adversarial training, the robustness and strength of the network can be increased. The following methods have been proposed for attacks:

- Fast Gradient Sign Method (FGSM) [20], [21]:
- Projected Gradient Descent (PGD) [22]
- Carlini and Wagner Attack (C&W) [23]
- One-pixel attack [24]
- Basic Iterative Method (BIM) [21]
- DeepFool [25]

Fast Gradient Sign Method Algorithm

One of the effective methods proposed for adversarial training is the FGSM [20], which optimally calculates an adversarial manipulation for an image.

This method is part of the white-box attacks because the adversary needs access to the architecture and parameters of the model at all times.

Equation: The equation for the FGSM algorithm is as follows:

$$X^{adv} = X + \epsilon \cdot sign(\nabla_X J(X, Y_{true})) \tag{1}$$

In equation (1), we have:

- X: the original and unchanged image
- X^{adv}: the adversarial example that is obtained after applying the FGSM to the original image
- ϵ: a constant to determine the severity of the attack (the size of the adversarial perturbation)
- J: the loss function
- Y_{true}: the actual class and label of the original image
- ∇_X: the gradient of the cost function

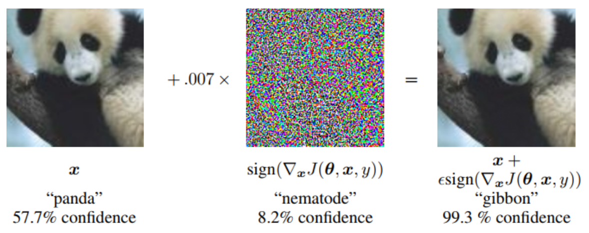

$+ .007 \times$
=
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Fig. 4. An example of applying the FGSM algorithm [20]

In this equation (1), the larger the value of ϵ, the more the adversarial example deviates from the original image.

How the algorithm works?: The goal of this method is to add a calculated noise, which is not random and is in the direction of the cost function's slope in the original image. In this method, the attacker exactly benefits from the method used to train the network in detecting classification boundaries. This is done by manipulating the modified image so that the loss function is directed towards taking another image wrong.

The FGSM is the best attack according to the L^{∞} rule for linear models, and since neural networks are non-linear models, it can be concluded that it does not work very well against neural networks.

An Example of Adversarial Sample: As shown in fig. 4, an image from the ImageNet dataset is presented, which has been trained on the GoogLeNet network. The network confidently identifies it as a panda with 57.7% certainty. Then, by adding a very small vector to the image, the network's classification can be thrown into error. It can be seen that the model identified the manipulated image as a long-armed monkey with 99.3% confidence.

IV. Methodology

To make the generated adversarial examples more effective in fooling the network and more realistic from a human perspective, a network called Adv-GAN [26] is used.

Problem Description

Assume that $X \subseteq R^n$ is the feature space, n is the number of features, (x_i, y_i) is i-th example in the training dataset, consisting of feature vectors,

- $x_i \in X$: Generated based on an unknown distribution $x_i \sim P_{data}$;
- $y_i \in Y$: The true and correct class label.

The goal of the learning system is to learn a classifier $f : X \rightarrow Y$ from the domain X to the set of classification outputs Y. More specifically, given a sample x, the goal of the attacker is to generate an adversarial example x_A such that $f(x_A) \neq y$ (in non-targeted attacks) or $f(x_A) = t$ (in targeted attacks), where t is the target class, and x_A is close to the original example x in terms of the L^2 distance or other distance metrics.

Adv-GAN

This network uses a feed-forward neural network to generate partial changes, and a discriminator network to guarantee the realism of generated samples. Compared to the FGSM, this approach allows for the immediate generation of adversarial examples for any input sample once the feed-forward network is trained. Therefore, without accessing the model itself (a semi-white-box attack) the adversarial sample can be generated. In addition, the Adv-GAN can be used for both white-box and black-box targeted and untargeted attacks.

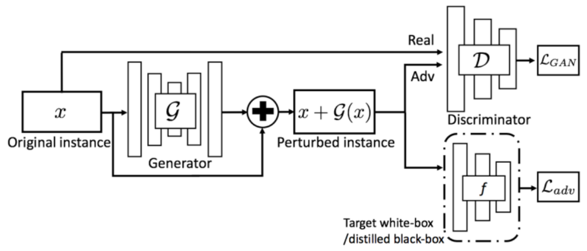


Fig. 5. The overall architecture of the Adv-GAN [26]

The overall architecture of the Adv-GAN is illustrated in fig. 5. The network consists of three neural networks:

1) The generator G
2) The discriminator D
3) The target neural network f

The generator takes the original sample x as input and produces a perturbation $G(x)$. Then $x + G(x)$ is fed to the discriminator D to distinguish between the generated sample and the original sample. Here, the role of D is to incentivize the generated samples to be indistinguishable from the original sample of that class. Finally, $x + G(x)$ is passed as input to f, which outputs its own loss L_{adv} according to equation 3 [26].

Equation 2 shows the adversarial loss:

$$L_{GAN} = \mathbb{E}_x log D(x) + \mathbb{E}x log(1 - D(x + G(x))) \quad (2)$$

Equation 3 is used to calculate the loss for fooling the target model f in a targeted attack:

$$L^f adv = \mathbb{E}x l_f(x + G(x), t) \quad (3)$$

It should be noted that the loss $L^f adv$ encourages the perturbed image to be misclassified into the target class t. To limit the amound of perturbation, a *Soft Hinge* loss on the L_2 norm is used in equation 4 where c is a user-specified bound:

$$L_{hinge} = \mathbb{E}_x \, max(0, \|G(x)\|_2 - c) \quad (4)$$

Therefore, the final objective (loss function) is given in equation 5:

$$L = L^f_{adv} + \alpha L_{GAN} + \beta L_{hinge} \quad (5)$$

where the parameters α and β correspond to the importance of each objective.

Finally, D and G are obtained by solving a min-max game presented in equation 6.

$$arg \; min_G \; max_D \; L \quad (6)$$

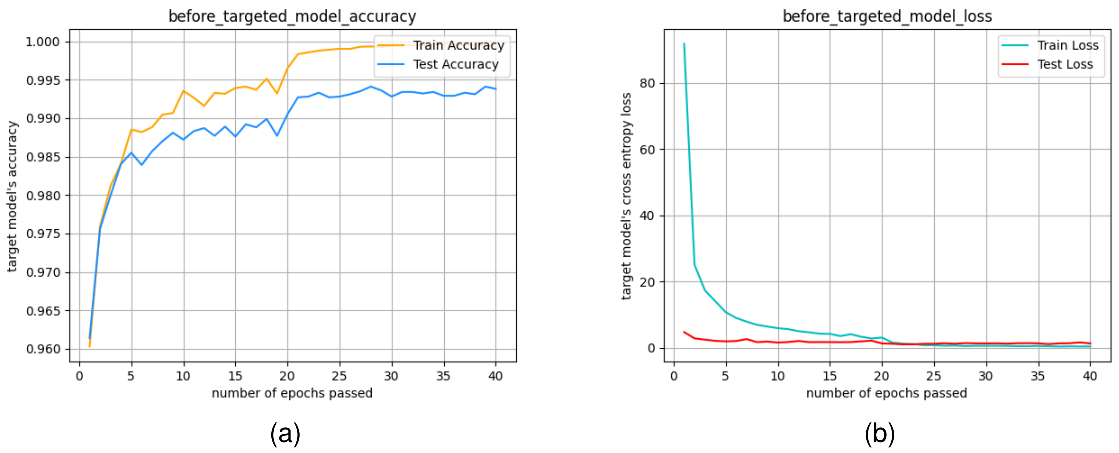


Fig. 6. The accuracy and loss of the target network before the attack.

V. Results & Discussion

The codes were run on a GPU.1080Ti.xlarge with 31.3 GB of RAM and 6 virtual CPUs, running Ubuntu 18.04. The MNIST dataset, consisting of 60,000 28*28 pixel images for training the network and 10,000 28*28 pixel images for testing, were used. The hyper-parameters for training the target network are set to 60 epochs and a batch size of 128 (fig. 11).

The generator network consists of three parts, including an encoder, bottleneck, and decoder. The encoder's architecture is shown in Fig. 12, which is followed by the bottleneck part consisting of four identical Res-Net blocks whose architecture is shown in Fig. 13. The final part of the generator is the decoder, shown in Fig. 14.

The discriminator network architecture is shown in Fig. 15.

Target Network Before the Attack

The performance of the target network before the attack is shown in Fig. 6a and Fig. 6b, where the accuracy and loss of the network are plotted against the training epochs. The network's accuracy and loss improve as the number of training epochs increases.

Finally, after the completion of the target network training, its performance is evaluated on a test set containing 10,000 samples. The network classifies 70 samples incorrectly and accurately identifies 9930 samples (99.3% accuracy) with a loss of 1.504620.

Adv-GAN

Fig. 7a shows the loss for the generator and discriminator of the Adv-GAN network. As expected, the generator improves in generating more natural samples (loss closer to 1) and the discriminator performs better in distinguishing fake/generated samples from real/natural ones (loss closer to 0) as the network undergoes more training epochs.

In Fig. 7b, the loss for the perturbation applied to the original image to generate adversarial samples is shown. As expected, the Adv-GAN network produces adversarial samples with less perturbation as it undergoes more training epochs, and thus, the generated samples become closer and more similar (loss closer to 0) to natural ones.

Fig. 7c shows the adversarial loss, which is the main objective of the network to minimize. It can be observed that as the network undergoes more training epochs, adversarial samples can deceive the target network more and more and be classified into different categories from the original image. This loss actually shows the performance of the Adv-GAN network in fooling the target network, and the closer this loss is to 0, the weaker the target network becomes against attacks.

Target Network After the Attack

After training the Adv-GAN and providing the adversarial examples generated by its generator were fed to the target model, the results of the target network on the test dataset in Fig. 8. The success rate of the attack is higher than 99.5%, which means that the target network, which previously failed to correctly classify 70 cases in the testing dataset, now correctly classifies only about 43 cases.

Adversarial Examples and Target Model

After training the Adv-GAN network, the generator can be used separately which gets a random noise sample and produces an adversarial example.

In Fig. 9, adversarial examples are shown for different original and predicted labels. In these figures, the rows indicate the original and true labels, and the columns indicate the predicted labels by the target model. Note that the empty spaces exist because the target model may never misclassify an image with the true label of 0 as the class 6, hence no adversarial example is generated for this case.

Fig. 10 shows the confusion matrices obtained for the generated adversarial examples from test data, respectively. In these matrices, rows represent the true labels and columns represent the predicted labels by the target model. The diagonal elements of the matrix indicate the number of correct predictions for each label and class. As can be seen, most of the adversarial examples were classified into a wrong class. Additionally, it

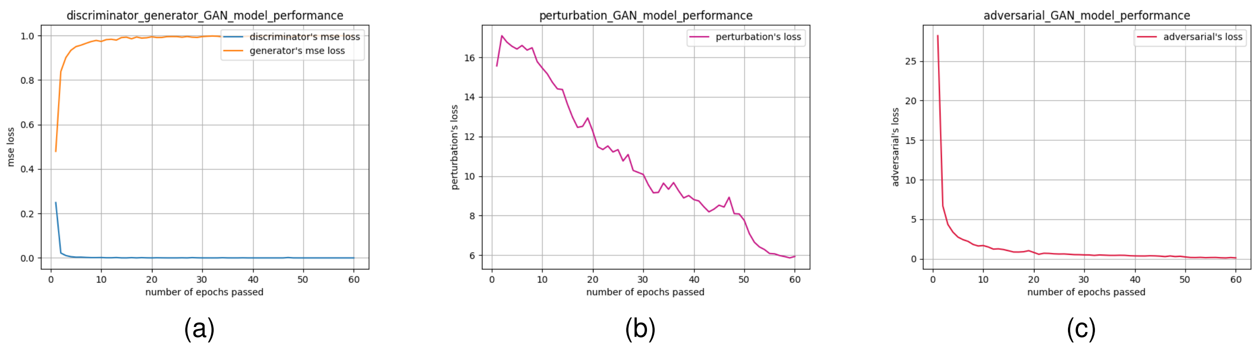
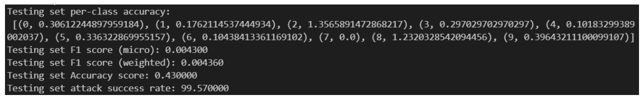


Fig. 7. The performance of the Adv-GAN while training.



```
Testing set per-class accuracy:
 [(0, 0.30612244897959184), (1, 0.1762114537444934), (2, 1.3565891472868217), (3, 0.297029702970297), (4, 0.10183299389
002037), (5, 0.336322869955157), (6, 0.10438413361169102), (7, 0.0), (8, 1.2320328542094456), (9, 0.39643211100099107)]
Testing set F1 score (micro): 0.004300
Testing set F1 score (weighted): 0.004360
Testing set Accuracy score: 0.430000
Testing set attack success rate: 99.570000
```

Fig. 8. Performance of the target network on the **testing** data after attack

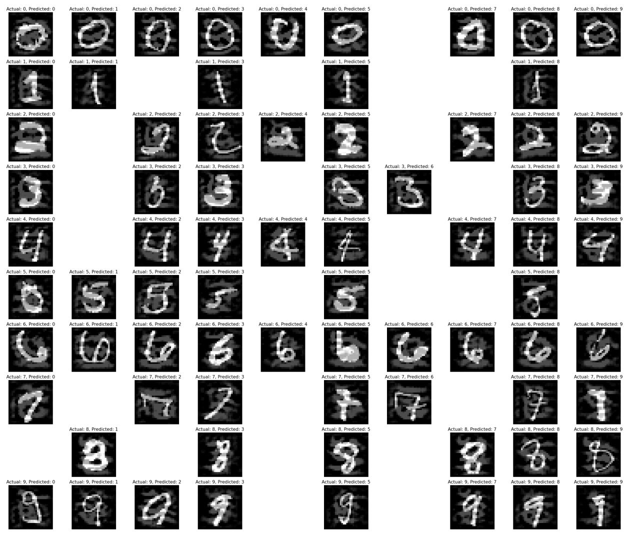
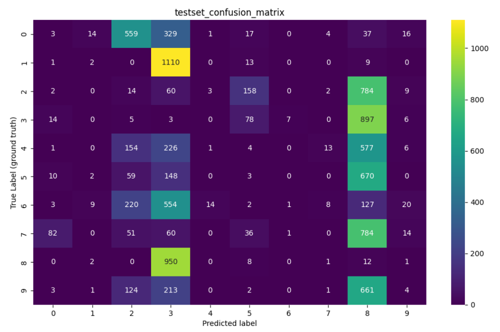


Fig. 9. Adversarial examples generated from test dataset images with original and predicted labels.

can be observed that, for example, most of the adversarial examples generated from class 1 training images were more similar to class 3 according to the target model.

VI. Conclusion & Future Work

In this project, the Adv-GAN method was examined as a novel and powerful attack method against DNNs. The main idea of this network is inspired by GANs. Therefore, this method can be used in gray-box and black-box attacks with a high attack success rate because when the generator part of the Adv-GAN network is trained, it can independently produce optimal adversarial manipulations. The adversarial examples generated by this method have very high real quality, and therefore, this method is a strong candidate for evaluating DNNs against adversarial examples. The trained network was able to reduce the performance of the target network from



	0	1	2	3	4	5	6	7	8	9
0	3	14	559	329	1	17	0	4	37	16
1	1	2	0	1110	0	13	0	0	9	0
2	2	0	14	60	3	158	0	2	784	9
3	14	0	5	3	0	78	7	0	897	6
4	1	0	154	226	1	4	0	13	577	6
5	10	2	59	148	0	3	0	0	670	0
6	3	9	220	554	14	2	1	8	127	20
7	82	0	51	60	0	36	1	0	784	14
8	0	2	0	950	0	8	0	1	12	1
9	3	1	124	213	0	2	0	1	661	4

Fig. 10. Confusion matrix for adversarial examples generated from test data images.

99.3% to a completely wrong performance of 0.43%. It is noteworthy that the existence of these adversarial examples shows that having high accuracy in explaining the dataset or even labeling it does not necessarily mean that the model has properly understood its task, and it is resistant to manipulated examples.

The vulnerability of neural networks to adversarial examples has led to very active research in the field of adversarial attacks and defense methods. In some research, techniques for defending neural networks against known attack methods are proposed, and in other areas, more powerful attack methods are being designed and presented. In some research, GANs have been used for both attacking and defending purposes. This extensive activity and effort will make DL methods much more resistant in the field of security and safety applications in the real world, and attackers will not be able to interfere and disrupt the network easily. Therefore, examining the performance of these attack methods on more complex and diverse datasets such as CIFAR-100 and ImageNet, as well as applying these attack methods on Recurrent networks and non-image textual datasets, is another step towards more reliable networks.

References

[1] M. Helmstaedter, K. Briggman, S. C. Turaga, V. Jain, H. Seung, and W. Denk, "Connectomic reconstruction of the inner plexiform layer in the mouse retina," *Nature*, vol. 500, pp. 168–174, 2013.

[2] H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, D. Merico, R. Yuen, Y. Hua, S. Gueroussov, H. Najafabadi, T. Hughes, Q. Morris, Y. Barash, A. Krainer, N. Jojic, S. Scherer, B. Blencowe, and B. Frey, “The human splicing code reveals new insights into the genetic determinants of disease,” *Science*, vol. 347, 2015.

[3] G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, “Deep neural networks for acoustic modeling in speech recognition,” *IEEE Signal Processing Magazine*, vol. 29, p. 82, 2012.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *NIPS*, 2014.

[5] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

[6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[7] A. Vedaldi and K. Lenc, “Matconvnet: Convolutional neural networks for matlab,” *Proceedings of the 23rd ACM international conference on Multimedia*, 2015.

[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *Proceedings of the 22nd ACM international conference on Multimedia*, 2014.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *ArXiv*, vol. abs/1603.04467, 2016.

[10] M. M. Najafabadi, F. Villanustre, T. Khoshgoftaar, N. Seliya, R. Wald, and E. A. Muharemagic, “Deep learning applications and challenges in big data analytics,” *Journal of Big Data*, vol. 2, pp. 1–21, 2014.

[11] N. Papernot, P. Mcdaniel, A. Sinha, and M. P. Wellman, “Towards the science of security and privacy in machine learning,” *ArXiv*, vol. abs/1611.03814, 2016.

[12] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. Mcdaniel, “Adversarial examples for malware detection,” in *ESORICS*, 2017.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, 2015.

[14] A. Giusti, J. Guzzi, D. Ciresan, F. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. Gambardella, “A machine learning approach to visual perception of forest trails for mobile robots,” *IEEE Robotics and Automation Letters*, vol. 1, pp. 661–667, 2016.

[15] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” *ArXiv*, vol. abs/1706.06083, 2018.

[16] N. Akhtar and A. Mian, “Threat of adversarial attacks on deep learning in computer vision: A survey,” *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.

[17] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song, “Robust physical-world attacks on deep learning models,” *arXiv: Cryptography and Security*, 2017.

[18] D. Heaven. Why deep-learning ais are so easy to fool. [Online]. Available: https://www.nature.com/articles/d41586-019-03013-5

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, “Generative adversarial nets,” in *NIPS*, 2014.

[20] I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *CoRR*, vol. abs/1412.6572, 2015.

[21] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *ArXiv*, vol. abs/1607.02533, 2017.

[22] ——, “Adversarial machine learning at scale,” *ArXiv*, vol. abs/1611.01236, 2017.

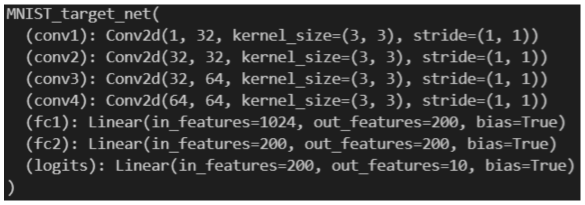
[23] N. Carlini and D. A. Wagner, “Towards evaluating the robustness of neural networks,” *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, 2017.

[24] J. Su, D. V. Vargas, and K. Sakurai, “One pixel attack for fooling deep neural networks,” *IEEE Transactions on Evolutionary Computation*, vol. 23, pp. 828–841, 2019.

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

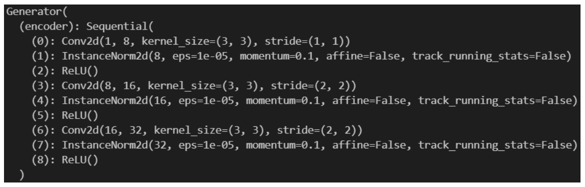
[26] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, “Generating adversarial examples with adversarial networks,” *ArXiv*, vol. abs/1801.02610, 2018.

Appendix
Architecture of Neural Networks



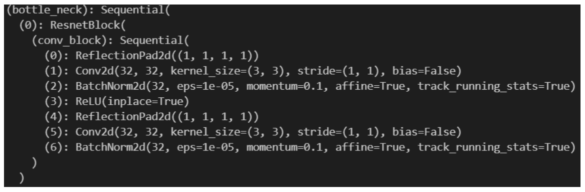
```
MNIST_target_net(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=1024, out_features=200, bias=True)
  (fc2): Linear(in_features=200, out_features=200, bias=True)
  (logits): Linear(in_features=200, out_features=10, bias=True)
)
```

Fig. 11. Architecture of target neural network



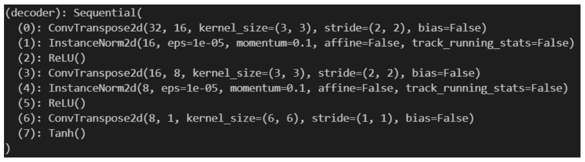
```
Generator(
  (encoder): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1))
    (1): InstanceNorm2d(8, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (2): ReLU()
    (3): Conv2d(8, 16, kernel_size=(3, 3), stride=(2, 2))
    (4): InstanceNorm2d(16, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (5): ReLU()
    (6): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2))
    (7): InstanceNorm2d(32, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
    (8): ReLU()
  )
```

Fig. 12. Architecture of the encoder in generator network



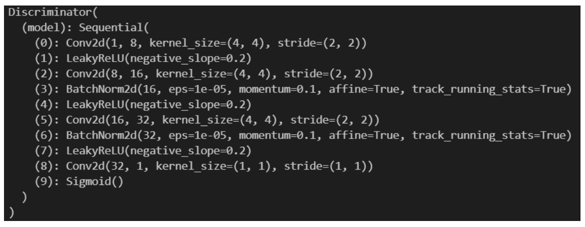
```
(bottle_neck): Sequential(
  (0): ResnetBlock(
    (conv_block): Sequential(
      (0): ReflectionPad2d((1, 1, 1, 1))
      (1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
      (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (3): ReLU(inplace=True)
      (4): ReflectionPad2d((1, 1, 1, 1))
      (5): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
      (6): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

Fig. 13. Architecture of the bottleneck in generator network



```
(decoder): Sequential(
  (0): ConvTranspose2d(32, 16, kernel_size=(3, 3), stride=(2, 2), bias=False)
  (1): InstanceNorm2d(16, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
  (2): ReLU()
  (3): ConvTranspose2d(16, 8, kernel_size=(3, 3), stride=(2, 2), bias=False)
  (4): InstanceNorm2d(8, eps=1e-05, momentum=0.1, affine=False, track_running_stats=False)
  (5): ReLU()
  (6): ConvTranspose2d(8, 1, kernel_size=(6, 6), stride=(1, 1), bias=False)
  (7): Tanh()
)
```

Fig. 14. Architecture of the decoder in generator network



```
Discriminator(
  (model): Sequential(
    (0): Conv2d(1, 8, kernel_size=(4, 4), stride=(2, 2))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Conv2d(8, 16, kernel_size=(4, 4), stride=(2, 2))
    (3): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2)
    (5): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2))
    (6): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2)
    (8): Conv2d(32, 1, kernel_size=(1, 1), stride=(1, 1))
    (9): Sigmoid()
  )
)
```

Fig. 15. Architecture of the discriminator network