METHODS TO DEFEND ADVERSARIAL IMAGES

Yifeng Xiong
Department of Computer Science
University of California, Irvine
Irvine, CA 92697, USA
yifengx4@uci.edu

ABSTRACT

Deep neural networks, such as CNN, ResNet, are useful tools for image identification. However, with some little change, the adversarial images could fool the neural network. In this project, I designed two models to defend those attacks: a preprocessing model based on Autoencoder, and a detector based on Normalizing flow. The dataset I use is CIFAR-10Krizhevsky (2009). For the preprocessing model, I first train an Autoencoder with clean images. Then I preprocess the image with the Autoencoder. I get 55% correct on the perturbed dataset. For the detector, I use a pre-trained Normalizing flow to get the latent value of the image, and I use a 5 layers fully connected neural network to classify whether it's clean or perturbed. I get 80% correct on the test dataset.

1 INTRODUCTION

Deep neural networks are the most common techniques that used for image identification. Residual Neural network, He et al. (2016), has test error 8.75% with 20 layers. However, these strong neural networks are not robust for some attack methods, such as Fast Gradient Sign Method (Goodfellow et al., 2014), Projected Gradient Descent (Madry et al., 2017), etc. Adversarial attacks can fool the neural networks with only small perturbations to images but cannot fool human beings. These attacks are often used in spam emails, where images can be easily modified to fool monitoring models. The main purpose for this project is to find some ways to enhance the robustness of neural networks. The advantage of my model is the high recognition accuracy. I generate perturbed images with three powerful attacks. After preprocessing with my autoencoder, the neural network get 52% correct on the perturbed dataset, and for the detector based on Normalizing flow, I get 80% correct on the test dataset. Even though those modified pictures have a small distance from the real picture, in the latent space, I believe the distance is larger. My second model confirmed what I thought. I use Normalizing flow to get the value of these images in the latent space, and then through a very simple fully connected neural network, I can get 80% recognition accuracy (classify whether the image is attacked).

2 RELATED WORK

2.1 PCA DETECTOR

Principal Component Analysis (PCA) projects a set of n-dimensional data points into a lower dimensional space where the data points exhibit the most variance and hence contain the most information. Hendrycks and Gimpel apply PCA to pairs of normal and adversarial images and find that the PCA coefficients for later principal components of adversarial images have larger variance compared to those of normal images (Hendrycks & Gimpel, 2016). The detector relies solely on the coefficient variance to tell if an image has been attacked. They try to map the data to a lower dimensional space, which is similar to my approach (mapping the data to the latent space.)

2.2 DISTRIBUTION DETECTOR

Grosse et al. (2017) compare the underlying distribution of normal images with the distribution of adversarial images and use a kernel-based two-sample test with Maximum Mean Discrepancy

Table 1: ResBlock structure (input: I: in channels, O: out channels, S: stride)

LAYER	DETAILS (bias is always false)
conv1	I, O, kernel size = 3, stride = stride, padding = 1
conv2	O, O, kernel size = 3, stride = 1, padding = 1
shortcut ($I = O$)	Identity
shortcut ($I \neq O$)	conv2d(I, O, kernel size = 1, stride = stride)

(MMD) and Energy Distance (ED) as statistical metric to flag adversarial behavior in a batch of images. However, the major drawback of this statistical approach is that it cannot detect adversarial examples on a per-input basis; even worse, the confidence of the test depends heavily on the number of samples in the image batch. My method avoids this problem.

2.3 Separate Neural Network

Gong et al. (2017) propose a simple binary classifier to separate adversarial examples from normal images trained on a mixture of normal and adversarial examples. The model successfully defends against FGSM and JSMA attacks with near 100% accuracy. My method also use a separate neural network to detect the perturbed image.

3 Dataset

The dataset I use in this project is CIFAR-10 (Krizhevsky, 2009). It contains 60000 images in 10 classes, with shape [3, 32, 32]. Before training the ResNet-20, I preprocess the data by RandomCrop and RandomHorizontalFlip. Due to the GPU limit, I use a pretrained Normalizing flow (van Amersfoort, 2020). The author preprocess the data by RandomAffine and RandomHorizontalFlip. I use the Python package Foolbox (Rauber et al., 2017) to generate the perturbed image.

4 Approach

In this section, I will introduce several methods and models I use in this project, and I'll discuss the result in the next section.

4.1 Residual Neural Network

The model I use to classify the images, to attack and defend is the Residual Neural Network introduced by He et al. (2016). I mainly follow the structure in the paper, and build a Residual neural network. The first layer is a convolutional layer which has 16 filters of shape (3, 3), stride and padding are both 1. Then it contains three ResBlocks, where each block has two convolutional layers, and a shortcut layer. Finally, it has a linear layer with input channel is 256, output channel is 10. More details of ResBlock are provided in Table 1. I train the model with 200 epoches. I get 88.29% correct on test data.

4.2 Attack Method

The goal for the attack method is to find a perturbed image that looks similar to the origin one, where human beings believe it's the same, but for neural networks, it is completely different. For an input image x, neural network F, we want to generate the perturbed image x^*, where $F(x^*) \neq F(x)$, and $Distance(x^*, x)$ is small. We usually choose $L2$ distance. Figure 1 shows some examples of these attack method. In the following, we introduce three attack methods we use.

4.2.1 Fast Gradient Sign Method

Goodfellow et al. (2014) introduces Fast Gradient Sign Method to generate adversarial images by using the derivative of the loss function of the model. It perturbs the image in the direction of

Figure 1: Clean images and perturbed images. The images are list in the order: clean, FGSM, PGD, Deepfool.

derivative by magnitude ϵ.

$$x^* = x + \epsilon \cdot sign(\nabla \boldsymbol{x} L(\theta, x, y)) \tag{1}$$

Where x is the input, x^* is the perturbed image, ϵ is the perturb rate, and L is the loss function.

4.2.2 Projected Gradient Descent

Madry et al. (2017) introduces Projected Gradient Descent, which is an iterative version of FGSM. Instead of perturbing the image only once with magnitude ϵ, it changes the image many times with a smaller magnitude α. Different from the Basic Iterative Method (Kurakin et al., 2016), PGD firstly adds some noise to the image.

$$x^{t+1} = \Pi_{x+S}(x^t + \alpha \cdot sign(\nabla x L(\theta, x, y))) \tag{2}$$

Where Π is the projection function, which maintains the distance between perturbed image and origin image.

4.2.3 Deepfool Method

Moosavi-Dezfooli et al. (2016) introduces Deepfool Method. It views the multi-class classifier as several binary classifiers. To perturb the image, it first finds the distance from the origin image to the separating affine hyperplane.

$$\hat{l}(x_0) = \arg\min_{k \neq \hat{k}(x_0)} \frac{|f_k(x_0) - f_{\hat{k}(x_0)}(x_0)|}{\|w_k - w_{\hat{k}(x_0)}\|_2} \tag{3}$$

Then we can get the minimum perturbation:

$$r_*(x_0) = \frac{|f_{\hat{l}(x_0)}(x_0) - f_{\hat{k}(x_0)}(x_0)|}{\|w_{\hat{l}(x_0)} - w_{\hat{k}(x_0)}\|_2^2} (w_{\hat{l}(x_0) - w_{\hat{k}(x_0)}}) \tag{4}$$

Where $f(x) = \mathbf{W}^\top x + b$ is the affine classifier, and w_k is the k^{th} column of $\mathbf{W}$.

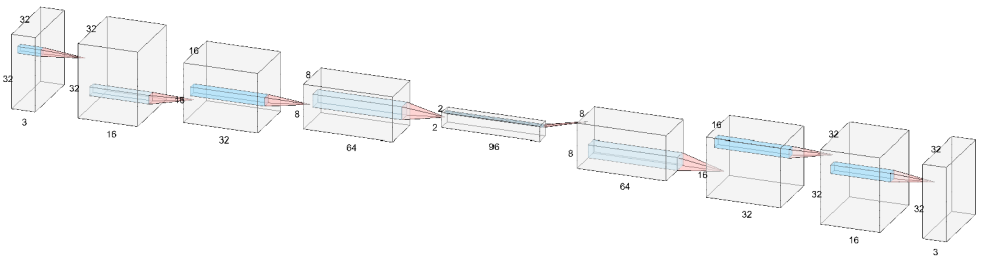

Figure 2: Structure of the Convolutional Autoencoder

4.3 Convolutional Autoencoder

Autoencoder can learn the main information of an image. It can remove the noise from the perturbed image. I train an autoencoder with only clean image. Instead of a traditional fully connected autoencoder, I use convolutional layers to decrease the dimension. It has four convolutional layers. Figure 2 is the structure of my autoencoder.

4.4 Normalizing flow and GLOW

Normalizing flow (Rezende & Mohamed, 2015) is an invertible mapping $f : \mathbb{R}^d \to \mathbb{R}^d$ between an arbitrary probability distribution and a simple probability distribution (usually Gaussian distribution). By the change of variable formula, we can write out the relation between these two probability distribution:

$$\rho_0(x) = \rho_1(f(x)) \cdot |\det \nabla f(x)| \tag{5}$$

where ρ_0 represents the arbitrary probability distribution and ρ_1 represents the Gaussian distribution. We can take $\log$ on both sides and get:

$$\log \rho_0(x) = \log \rho_1(f(x)) + \log |\det \nabla f(x)| \tag{6}$$

To train the Normalizing flow, we should maximize the expected log-likelihood, which is the right hand side in (6). However, computing the log-determinant in general requires $O(d^3)$. One of the ways to reduce the complexity is to make the Jacobian matrix triangular. There are many different designs, for examples, NICE (Dinh et al., 2014), Real NVP (Dinh et al., 2016). The model I use in this project is GLOW (Kingma & Dhariwal, 2018). There are three substeps in one step of flow in Glow: Actnorm, Invertible 1×1 convolution, and Affine Coupling Layers. The details are provided in Table 2.

GLOW is combined with a multi-scale architecture. Figure 3 is the full structure.

5 Results

5.1 Method 1: preprocess by Autoencoder

5.1.1 Summary of the model

Firstly, I trained the Autoencoder with the clean images. Then I use the Autoencoder to preprocess the data. I input the preprocessed data into the ResNet to classify their classes. Figure 4 is the flow chart of the model.

Table 2: One step of flow in the Glow model

LAYER	DETAILS
Actnorm	$\mathrm{z}_{i,j} = s \odot x_{i,j} + b$
Invertible 1×1 convolution	$\mathrm{z}_{i,j} = W \cdot x_{i,j}$
Affine Coupling layers	$\mathrm{x}_1, x_2 = split(x)$ $h = N(x_1)$ $s, t = split(h)$ $t = sigmoid(t + 2)$ $x_2 = x_2 + s$ $x_2 = x_2 \cdot t$ $z = concat(x_1, x_2)$

Table 3: Result of Method 1, overall: 3000 FGSM, 3000 PGD, 4000 DeepFool

ResNet Correct rate	5000 FGSM	5000 PGD	5000 DeepFool	Overall
Before Preprocess	0%	0%	0%	0%
After Preprocess	47.92%	53.62%	55.9%	52.39%

5.1.2 RESULT

All the tests below are using the test dataset, which is not used during training. The ResNet I trained has 88.29% correct rate. By setting the ϵ to 0.02, I get the the attack success rate as follows: FGSM 82.58%; PGD 99.99%; DeepFool 99.05%. I test several combinations with perturbed images. In summary, before preprocess, the correct rate for ResNet is 0%, after preprocess, the correct rate raised up to near 50%. More details are provided in Table 3.

5.2 METHOD 2: DETECTOR BY NORMALIZING FLOW AND FULLY CONNECTED NEURAL NETWORK

5.2.1 SUMMARY OF THE MODEL

Due to the limit of GPU I have, I use a pretrained GLOW model from van Amersfoort (2020). I use GLOW to generate the latent space value for the training data, which shape is (48, 4, 4), and then I use these value as input to train the 5 layers fully connected neural network, with output 0 (clean) or 1 (perturbed). Figure 5 is the flow chart of the model.

5.2.2 RESULT

All the tests below are using the test dataset, which is not used during training. The ResNet I trained has 88.29% correct rate. By setting the ϵ to 0.02, I get the the attack success rate as follows: FGSM 82.58%; PGD 99.99%; DeepFool 99.05%. I test several combinations with perturbed images. In summary, before preprocess, the correct rate for ResNet is 0%, after preprocess, the correct rate raised up to near 50%. More details are provided in Table 4.

5.3 COMPARISON WITH OTHER DEFEND METHODS

I compare my second model with other three models mentioned in related works. I use the data from their paper. One of the papers only test on MNIST, and all of them don't test on Deepfool. As a conclusion, my model get similar result on FGSM and PGD to them, what's more, my model also works on Deepfool, more detailed are provided in Table 5

Table 4: Result of Method 2, overall: 10000 Clean, 3000 FGSM, 3000 PGD, 4000 DeepFool

Test Dataset	Detector correct rate
5000 Clean 5000 FGSM	97.12%
5000 Clean 5000 PGD	97.07%
5000 Clean 5000 DeepFool	63.17%
Overall	83.605%

Table 5: Comparison between several models

Method	Dataset	FGSM	PGD	DeepFool
PCA Detector	CIFAR-10	100%	92.8%	N/A
Distribution Detector	MNIST	99.78%	N/A	N/A
Separate Neural Network	CIFAR-10	100%	N/A	N/A
My Detector	CIFAR-10	97.12%	97.07%	63.17%

6 Conclusions and discussions

In this project, I design two methods to defend the adversarial images. The first one is a preprocessing method based on Autoencoder. I input the preprocessed data into the ResNet, and get near 50% correct, while with out the model, I get 0% correct. The second one is a detector based on Normalizing flow. I use the Normalizing flow to map the data to the latent space, and use a simple fully connected neural network to detect whether it is clean or perturbed. I get over 97% correct on FGSM and PGD, and 63% on DeepFool. There are many parts I could improve, for example, I can use more attack methods. And I can use other datasets like CIFAR-100 and ImageNet. I can also use other Normalizing flow, for example, the continuous normalizing flow.

7 Acknowledging section

I work alone on this project. In this project, most of the code was written by myself. Due to the GPU limit, I cannot train the GLOW locally, so I use the code from van Amersfoort (2020), and their pretrained model. I would like to thank Boyang Huang from University of Michigan, Ann Arbor. He provides many suggestions to this project.

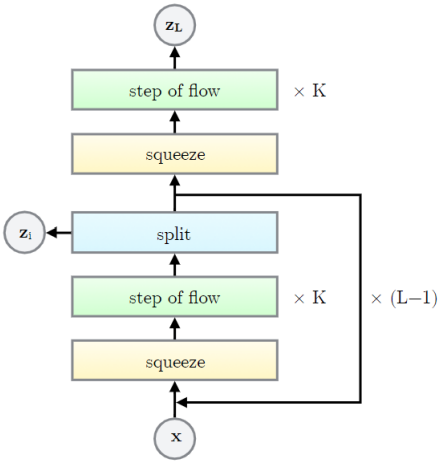

Figure 3: GLOW structure (Image source: Kingma & Dhariwal (2018))

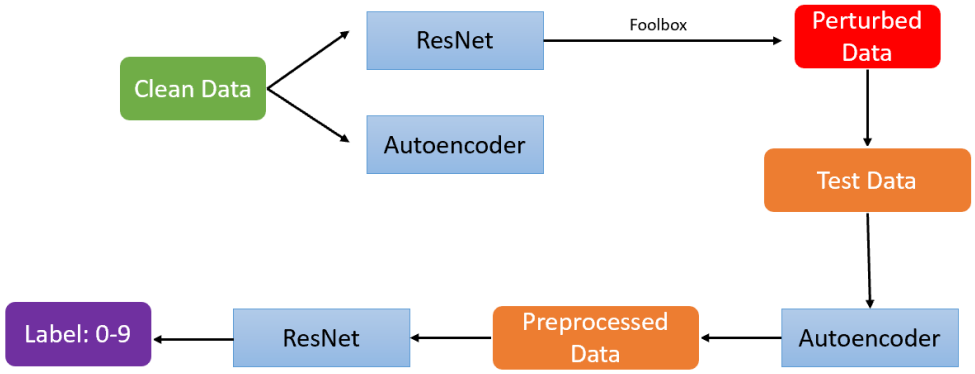

Figure 4: Flow chart, output is the class ResNet classified

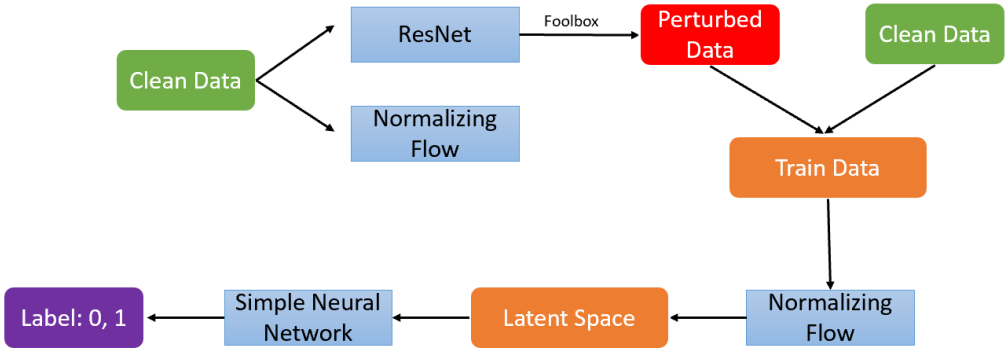

Figure 5: Flow chart, output is clean or not

REFERENCES

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. Adversarial and clean data are not twins. *arXiv preprint arXiv:1704.04960*, 2017.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. June 2016.

Dan Hendrycks and Kevin Gimpel. Early methods for detecting adversarial images. *arXiv preprint arXiv:1608.00530*, 2016.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. Adversarial examples in the physical world, 2016.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. pp. 2574–2582, 2016.

Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. In *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017. URL `http://arxiv.org/abs/1707.04131`.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pp. 1530–1538. PMLR, 2015.

Joost van Amersfoort. Glow-pytorch, 2020. URL `https://github.com/y0ast/Glow-PyTorch`.