Using Regularization Methods to Tackle Adversarial Images Classification Problems

Linchen Deng, Yimo Zhang, Yan Zhao

Brown University

This project is submitted for the requirement of

CSCI 1470: Final Project Evaluation

December 2019

Table of contents

1 Introduction **1**

2 Methodology **3**

2.1 Generating Adversarial Examples 4

2.1.1 Fast Gradient Sign method(FGS) 4

2.1.2 DeepFool 4

2.2 Model Training 6

2.2.1 Basic Model Architecture 6

2.2.2 Fine-tuning MLP 6

2.2.3 Deep Defense 7

2.2.4 Vanilla MLP 7

2.3 Evaluation Metrics 7

3 Results **11**

3.1 Image Comparison 11

3.2 Accuracy and ρ_2 12

3.3 ROC curve 15

4 More About this Project **19**

4.1 Challenges 19

4.2 Reflection 19

4.3 Github code 20

References **21**

Chapter 1

Introduction

The paper we chose is called "Deep Defense: Training DNNs with Improved Adversarial Robustness" [3]. The objective of the paper is to come up with a solution that improves deep neural networks' resistance and robustness against adversarial images. In the real-world, it remains a problem that well-trained deep neural networks don't generalize well for the adversarial images. Therefore, We are interested in finding solutions to address these problems in computer vision. In essence, this is an image classification problem.

In this project, we used the MNIST-Fashion dataset, which contains ten labelled classes on 60000 training and 10000 testing images. Using the pretrained Multi-Layer Perceptron(MLP) model, we generated adversarial images based on the algorithms proposed in paper DeepFool[2] and fast gradient sign method(FGS) [1]. Following the Deep Defense paper, we continued to train our MLP on the DeepFool images with and without regularization term in the loss function. After training, we evaluated the robustness of our modified models over both benign and FGS images. We concluded that the Deep Defense method improved the robustness of the model on FGS adversarial images.

Chapter 2

Methodology

We are basically comparing the performance of three models. The first model was trained on benign training images with regular cross entropy loss function for 15 epochs; the second model was fed with the pre-trained parameters from the first model after 10 epochs and was trained on DeepFool attacked training images with the same loss function for 5 extra epochs; the thrid model also took the parameters from the first model after 10 epochs and was then trained on the DeepFool attacked training images with a regularized loss function for 5 epochs.

After training the three models, we used them on both benign test images and test images attacked by FGS and obtained the accuracy. Moreover, the ρ_2 value was calculated for evaluating the model robustness.

In this part, we first shortly described the methods used to generate adversarial example; then clarified the model structure (e.g hyperparameters).

2.1 Generating Adversarial Examples

2.1.1 Fast Gradient Sign method(FGS)

The algorithm FGS proposed by [1] is as follows:

$$\hat{\boldsymbol{r}_i} = \epsilon \cdot \text{sign}(\nabla_{\boldsymbol{x}_i} L(\boldsymbol{x}_i, y_i, \boldsymbol{W})),$$

where $(\boldsymbol{x}_i, y_i)$ refers to training samples; $L(\cdot)$ refers to loss function, $\boldsymbol{W}$ refers to all learnable parameters (i.e weights and bias) nad $\hat{\boldsymbol{r}_i}$ refers to obtained noise added to $\boldsymbol{x}_i$. The intuition behind this method is quite simple: the sign of gradient of loss with respect to each image $\boldsymbol{x}_i$ indicates the direction of changes on $\boldsymbol{x}_i$ in order for increasing L. Then adding the scaled term sign of gradient would increase the loss function. Therefore for those FGS attacked images, the loss function is no longer minimized, and the model would be likely to misclassify them.

2.1.2 DeepFool

The idea of DeepFool [2] is more straightforward compared to FGS. The objective of DeepFool algorithm is to obtain the minimal perturbation to change the model's prediction on each traning sample:

$$\hat{\boldsymbol{r}_i} = \arg\min_{\boldsymbol{r}_i} \|\boldsymbol{r}_i\|_2$$
$$\text{subject to } f(\boldsymbol{x}_i) \neq f(\boldsymbol{x}_i + \boldsymbol{r}_i),$$

where $\boldsymbol{r}_i$ refers to the noise added to $\boldsymbol{x}_i$ and $f(\cdot)$ refers to the predicted class of a certain sample.

The algorithm for calculating DeepFool noise is the core of the DeepFool paper, while understanding and implementing this algorithm is one of the challenges of this project,

it's also the most time-consuming part of this part. Here we demonstrate the DeepFool algorithm. Assume we have logit $g_k(\cdot)$ for each of the class and the classification is made by chooseing the index associated with the largest logit (in our case, the number of cases is 10):

$$\boldsymbol{g}(\boldsymbol{x}_i) = (g_1(\boldsymbol{x}_i), g_2(\boldsymbol{x}_i), ..., g_{10}(\boldsymbol{x}_i))$$

$$f(\boldsymbol{x}_i) = \arg\max_k g_k(\boldsymbol{x}_i),$$

in a deep network consisting of dense layers, the logits $g_k(\cdot)$ are calculated by multiplying the weights and inputs and adding bias with ReLU activation, without applying the softmax function.

Algorithm 1 DeepFool Algorithm

inputs: $\boldsymbol{x}$, logits function: $g_k(\cdot)$
output: the minimal perturbation $\hat{\boldsymbol{r}}$

Initialize $\boldsymbol{x}_0 = \boldsymbol{x}$, $i = 0$, $\hat{\boldsymbol{r}} = \boldsymbol{0}$, **max_iter** $= 5$, **iter** $= 0$
while $f(\boldsymbol{x}_i) = f(\boldsymbol{x}_0)$ & **iter** $\leqslant$ **max_iter** **do**
  for $k \neq f(\boldsymbol{x}_0)$ **do**
    $\boldsymbol{w}_k = \nabla g_k(\boldsymbol{x}_i) - \nabla g_{f(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
    $h_k = g_k(\boldsymbol{x}_i) - g_{f(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
  end for
  $l = \arg\min_{k \neq f(\boldsymbol{x}_0)} \frac{|h_k|}{\|\boldsymbol{w}_k\|_2}$
  $\boldsymbol{r}_i = \frac{|h_l|}{\|\boldsymbol{w}_l\|_2^2} \boldsymbol{w}_l$
  $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + 1.02\boldsymbol{r}_i$, $\hat{\boldsymbol{r}} = \hat{\boldsymbol{r}} + 1.02\boldsymbol{r}_i$
  $i = i + 1$, **iter** $=$ **iter** $+ 1$
end while

2.2 Model Training

2.2.1 Basic Model Architecture

We used Multi-Layer Perceptron model in this project. It takes input shape of [batch size $\times$ 784] and output the shape of [batch size $\times$ 10] with one hidden layer with 100 units. The batch size was set to be 64. ReLU activation was applied at the hidden layer. All the weights were initialized by truncated normal distribution of mean 0 and standard deviation 0.01. Adam optimizer was used with learning rate 1×10^{-3}. Loss was calculated using cross entropy.

The model described above was referred as vanilla MLP in later context. The vanilla MLP was trained on benign set (training images without noise) for 10 epochs. For later comparison, we wrote the cross entropy loss function as:

$$\sum_k L(y_k, f(\boldsymbol{x}_k; \boldsymbol{W})),$$

where $(\boldsymbol{x}_k, y_k)$ refers to training samples; $f(\cdot)$ refers to MLP model; $\boldsymbol{W}$ refers to all learnable parameters, which, in our case, includes weights and bias.

2.2.2 Fine-tuning MLP

The weight of fine-tuning model was initialized with the trained weights of vanilla MLP. Without modifying the loss function, this model was trained on DeepFool images for 5 more epochs.

2.2.3 Deep Defense

The weight of Deep Defense model was initialized with the trained weights of vanilla MLP. With model architecture unchanged, its objective loss function was modified to include two regularization term:

$$\min_{\boldsymbol{W}} \sum_k L(y_k, f(\boldsymbol{x}_k; \boldsymbol{W})) + \lambda \sum_{k \in \mathcal{T}} \exp\left(-c \frac{||\Delta_{\boldsymbol{x}_k}||_2}{||\boldsymbol{x}_k||_2}\right) + \lambda \sum_{k \in \mathcal{F}} \exp\left(d \frac{||\Delta_{\boldsymbol{x}_k}||_2}{||\boldsymbol{x}_k||_2}\right), \quad (2.1)$$

$||\boldsymbol{x}_k||_2$ is the Euclidean norm for vector $\boldsymbol{x}_k$; $\Delta_{\boldsymbol{x}_k}$ is the DeepFool perturbation calculated using 1 and $\|\Delta_{\boldsymbol{x}_k}\|_2$ refers to its norm. $\mathcal{F}$ is the index set of misclassified DeepFool training samples, $\mathcal{T}$ is its complement, $c; d > 0$ are two scaling factors that balance the importance of different samples. We used $c = 25, d = 5, \lambda = 15$ in our loss function. This model was trained on DeepFool images for 5 more epochs.

2.2.4 Vanilla MLP

For comparability in terms of epochs, the vanilla MLP was trained on the original dataset for 5 more epochs.

2.3 Evaluation Metrics

We used accuracy, ρ_2 score and ROC as evaluation metrics.

Firstly, to evaluate the model, we used accuracy to measure the model performance on benign images and FGS images. For accuracy on FGS images, we calculated the smallest ϵ such that 50% of the pertubated images are misclassified by the Deep Defense regularized model, and denoted as ϵ_{ref}. Then we reported the testing accuracy of the

vanilla MLP, Fine-tuning MLP, Deep Defense on FGS images with ϵ equals to ϵ_{ref}, $0.5\epsilon_{ref}$, and $0.2\epsilon_{ref}$ (see results section).

ρ_2 score was proposed by DeepFool paper [2] and was used to measure the robustness of models on DeepFool images, which represents the relative noise needed to generate in order for attacking a given model to the same extent. The higher the value, the more robust the model is. ρ_2 is defined as:

$$\rho_2 := \frac{1}{|\mathcal{D}|} \sum_{k \in \mathcal{D}} \frac{||\Delta_{x_k}||_2}{||x_k||_2}, \tag{2.2}$$

in which $\mathcal{D}$ is the test images with DeepFool perturbation.

The ROC curves was used to get a better comparison of classification performance of different models. It was widely used in binary-classification problem, where the output of a specific model is a probability, and each point on ROC is obtained by using a certain value as the threshold h such that probabilities larger than h is classified as one class.

$$f(\boldsymbol{x}_i) = \begin{cases} \text{class one} & p(\boldsymbol{x}_i) > h \\ \text{class two} & p(\boldsymbol{x}_i) \leqslant h \end{cases}.$$

The corresponding point on ROC for every h is defined as:

$$\left(1 - \frac{\sum_i I(f(\boldsymbol{x}_i) = \text{class two})}{\sum_i I(y_i = \text{class two})}, \ \frac{\sum_i I(f(\boldsymbol{x}_i) = \text{class one})}{\sum_i I(y_i = \text{class one})}\right).$$

Area under ROC curve is an important evaluation metric too, which measures the overall performance of a model for all threshold value from 0 to 1. The larger the area

is, the better the predictive performance of a model is.

In multi-class scenario, we can draw ROC curve and calculate area under ROC curve by looking at each of the class. Notice that the output of our Neural Network for each image is a vector of probabilities. Then for each class k, we sum the other probabilities as the probability "not being class k", then the obtained ROC curve measures the predictive performance of neural network on class k.

Chapter 3

Results

3.1 Image Comparison

A visual demonstration of the original and noised images with DeepFool and FGS at $\epsilon = 0.05$ is in Figure 1-3.

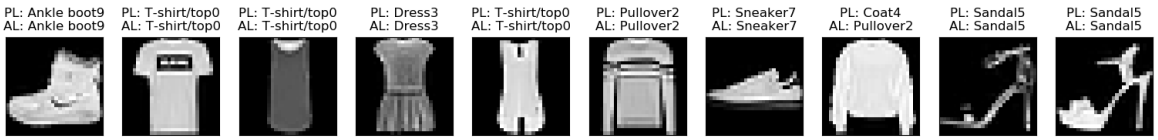

Fig. 3.1 Original Images without noise

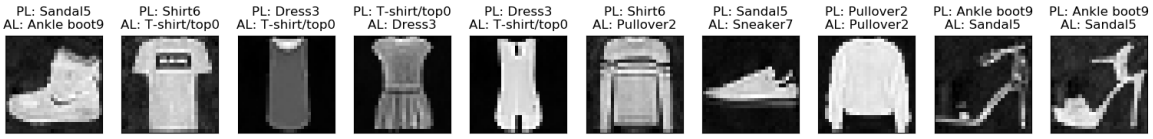

Fig. 3.2 Images with DeepFool noise

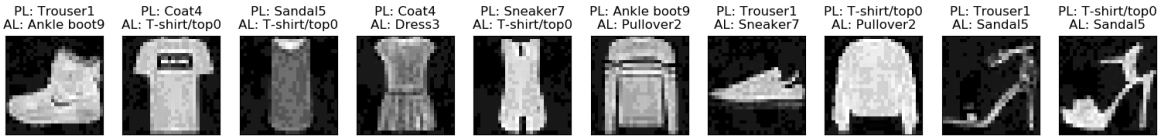

Fig. 3.3 Images with FGS noise at $\epsilon = 0.05$

As we can see from 3.2 and 3.3, both FGS and DeepFool are very strong tools for generating adversarial examples as both of them change the prediction of a model.

3.2 Accuracy and ρ_2

The results for accuracy and ρ_2 are shown below:

	Acc.	ρ2	Acc. @0.2Ɛref	Acc. @0.5Ɛref	Acc. @1.0Ɛref
Vanilla MLP	87.38%	3.92E-02	79.57%	63.26%	34.53%
Adversarial Training	88.13%	4.36E-02	81.50%	68.31%	42.97%
Deep Defense	87.99%	5.54E-02	82.74%	71.81%	50.88%

Fig. 3.4 Accuray and ρ_2 measurements. Column 1: accuracy on benign test set; Column $3-5$: accuracy on FGS attacked test set of different values of ϵ; Column 2: ρ_2 scores of the three models. The first row refers to the vanilla model; the second row refers to fine-tuning model; the third row refers to deep defense model.

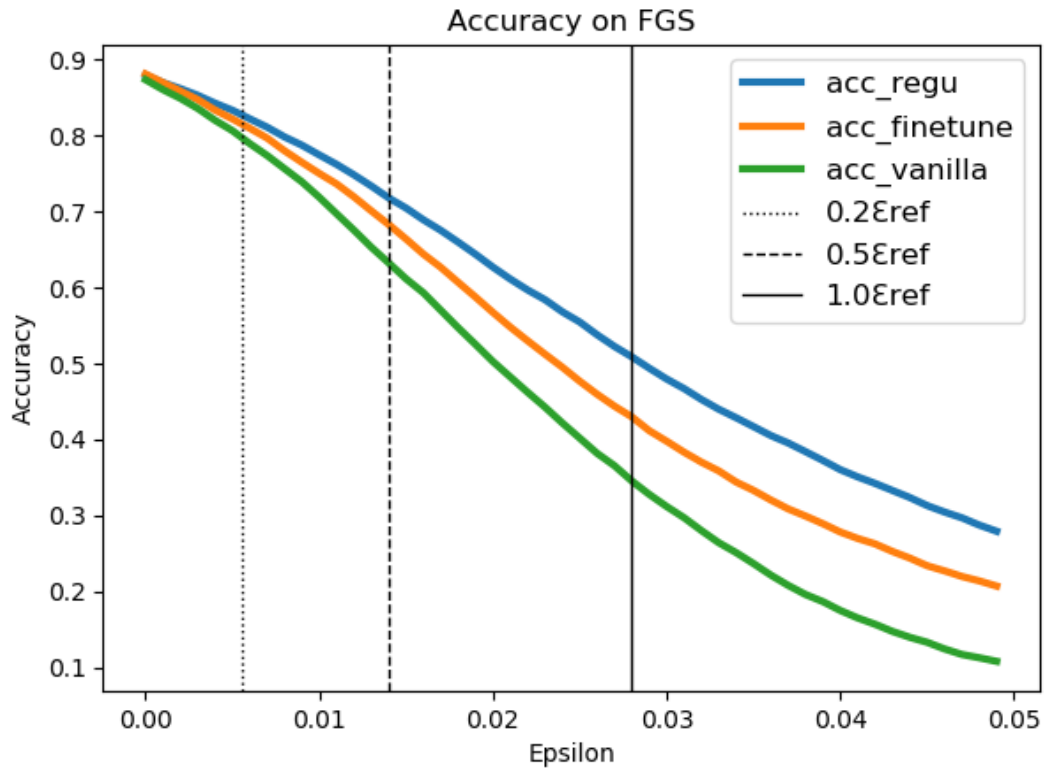

Fig. 3.5 Accuracy on FGS attacked images for different values of *epsilon*.

All models had similar accuracy on benign sets. The fine-tuning model had the highest accuracy (88.13%), Deep Defense had the second highest accuracy (87.99%), and the vanilla MLP had the lowest (87.38%).

The ρ_2 score was the highest in Deep Defense (5.54×10^{-2}), following by the fine-tuning model (4.36×10^{-2}) and the vanilla MLP (3.92×10^{-2}).

The testing accuracy on the FGS-pertubated images was the highest on Deep Defense, followed by the fine-tuning model and the vanilla MLP. The following figure 3.5 is more informative of accuracy on FGS attacked test images for different values of ϵ. The three black vertical lines indicates the value of ϵ_{ref} mentioned in 3.4; the blue curve

refers to the accuracy of Deep Defense model; the orange line refers to the accuracy of Fine-tunning model; the green lines refers to the accuracy of vanilla model.

3.3 ROC curve

	Vanilla MLP	Adversarial Training	Deep Defense
label 1	0.8534	0.9884	0.9878
label 2	0.9460	0.9995	0.9995
label 3	0.9606	0.9822	0.9817
label 4	0.9407	0.9941	0.9939
label 5	0.9335	0.9839	0.9843
label 6	0.9167	0.9994	0.9993
label 7	0.8074	0.9605	0.9587
label 8	0.9172	0.9986	0.9984
label 9	0.9323	0.9989	0.9989
label 10	0.9391	0.9988	0.9987

Fig. 3.6 Area Under the Curve(AUC) for each of the 10 classes.

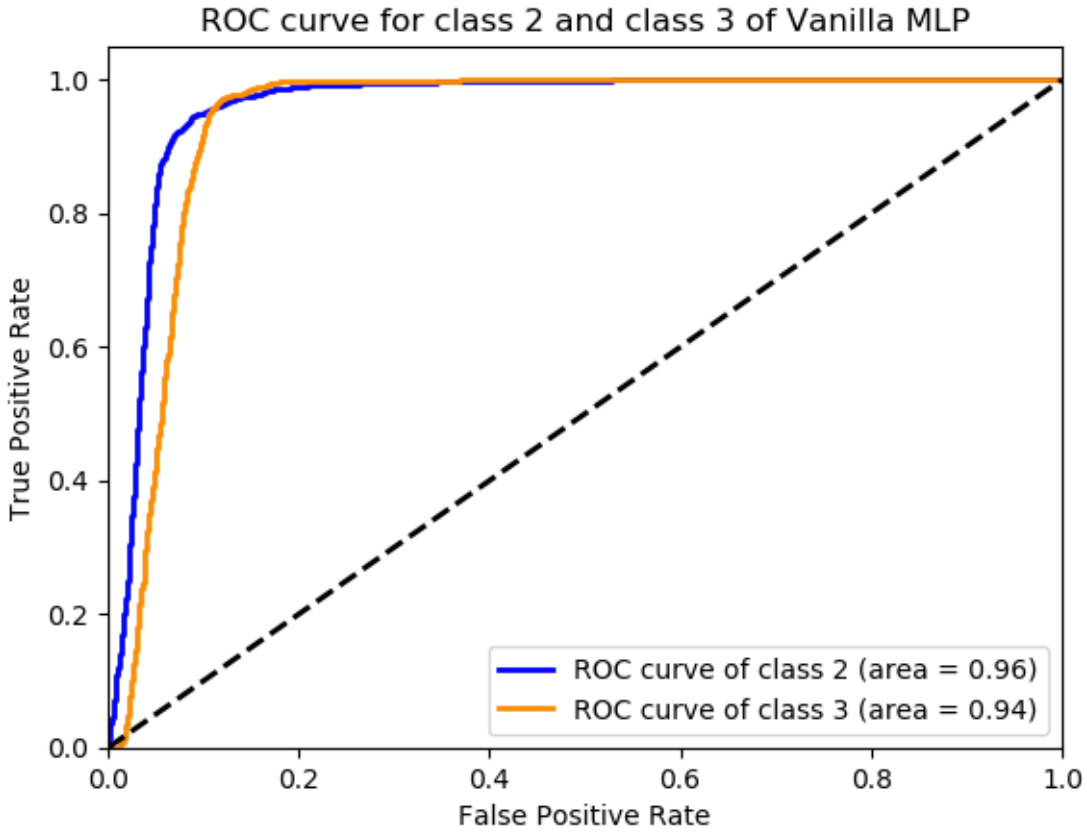

Fig. 3.7 ROC curve for class 2 and class 3 on vanilla model

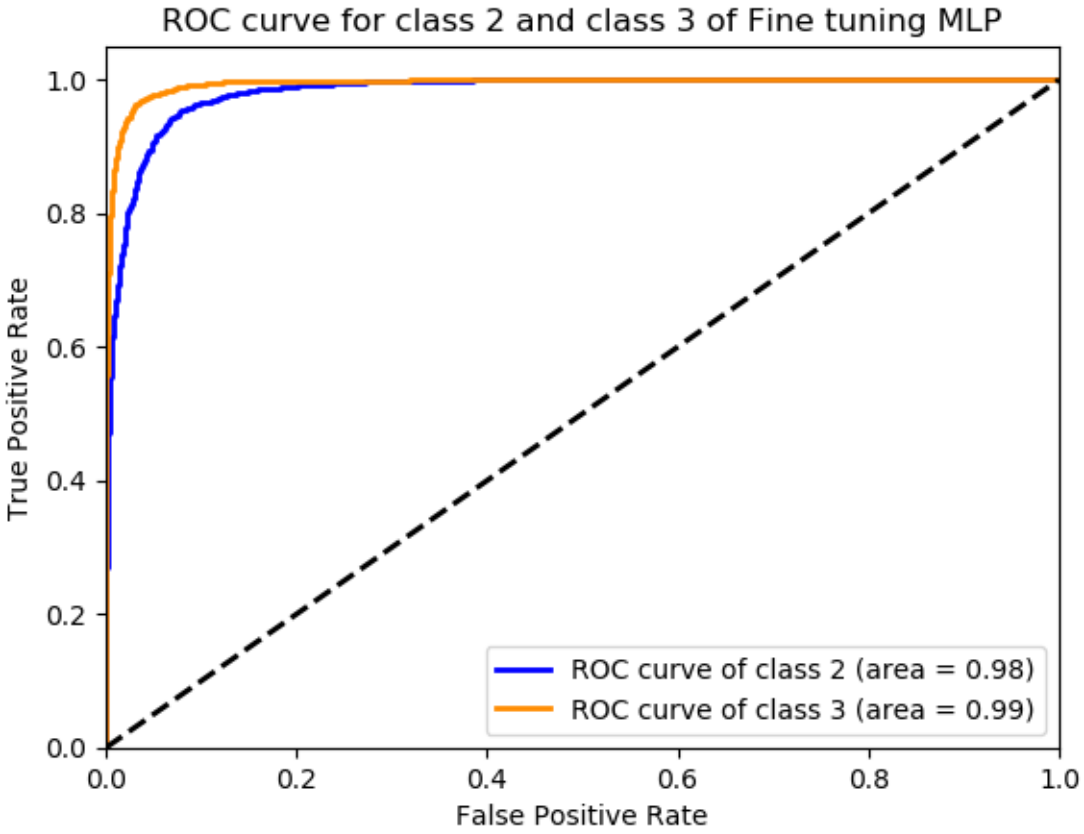

Fig. 3.8 ROC curve for class 2 and class 3 on fine tuning model

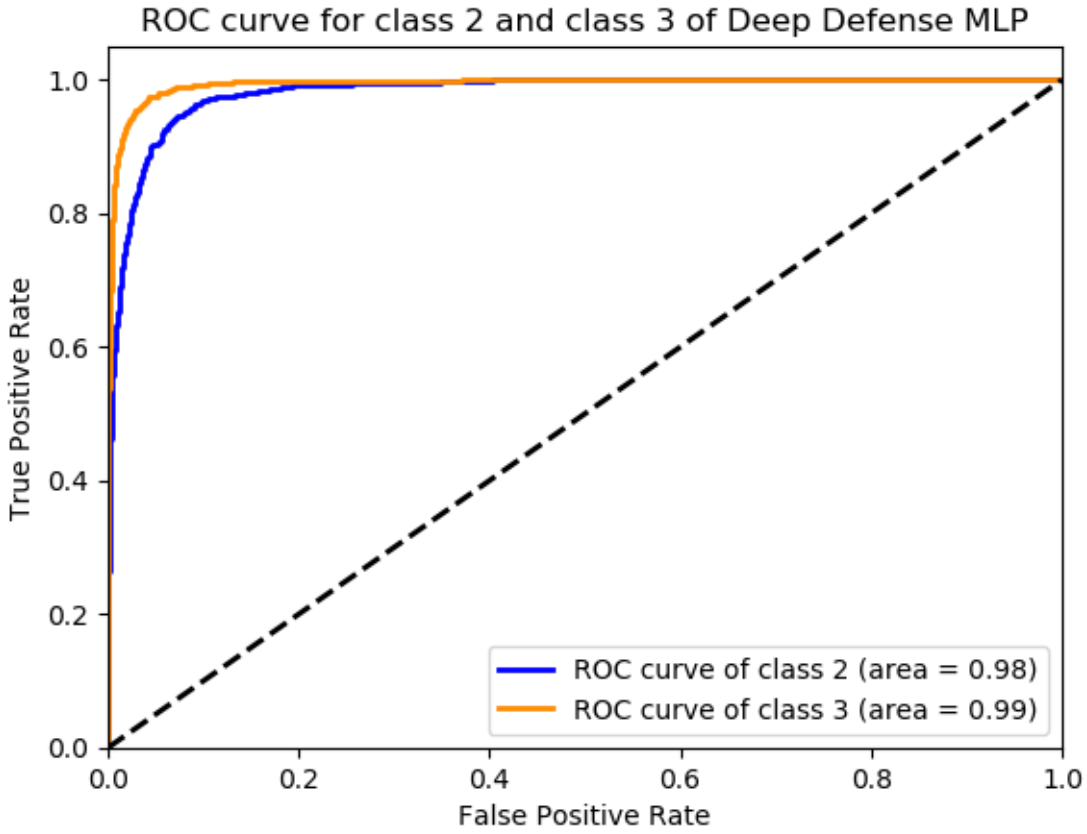

Fig. 3.9 ROC curve for class 2 and class 3 on regularized fine tuning model

For demonstration ability, ROC curve of only class 2 and 3 are shown. The curves were drawn based on DeepFool test data on the original vanilla MLP after its first training. The ROC curve for Vanilla model has higher false positive rate, especially for the class 3, the initial true positive rate is relatively low. Correspondingly, for the area under the ROC curve, vanilla model has the lowest ones which are 0.96 for class 2 and 0.94 for class 1. The full area under the curve can be seen

The ROC curve for fine tuning model and regularized fine tuning model are similar, which means both models have similar performance in classifying class 2 and class 3. Besides, the values for the area under the curve are the same for the two methods.

Chapter 4

More About this Project

4.1 Challenges

At the beginning of the project, we had some difficulty of understanding the DeepFool image generating methods the paper is explaining. Therefore, we referred to the DeepFool paper where it describes the algorithm clearly.

For the implementation of the regularization, although code was available on Github, we had trouble understanding its logic which is more than what the paper suggested, so we write our code in TensorFlow from scratch.

4.2 Reflection

Due to limited time, we did not spend much time on the parameter tuning for regularization term in Deep Defense model, which would have achieved higher testing accuracy on benign sets.

Moreover, in the Deep Defense paper [3], in addition to using MLP as deep neural network structure, it also used convolutional neural networks, which is more time-consuming to train and requires our more exploration on implementing the DeepFool algorithm (i.e get the gradient right). If had more time, we would also experiment on more deep neural network structures and more types of data.

4.3 Github code

https://github.com/Simonzym/Cool_Nerds

References

[1] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

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

[3] Yan, Z., Guo, Y., and Zhang, C. (2018). Deep defense: Training dnns with improved adversarial robustness. In *Advances in Neural Information Processing Systems*, pages 419–428.

References