# Robust Loss Functions: Defense Mechanisms for Deep Architectures

Trung-Kien Vu
Department of Computer Science
Faculty of Information Technology
VNU University of Engineering and Technology
Email: kienvu@vnu.edu.vn

Quoc-Long Tran
Department of Computer Science
Faculty of Information Technology
VNU University of Engineering and Technology
Email: tqlong@vnu.edu.vn

***Abstract*—Current deep learning methods and technologies have reached the level of deployment in softwares and hardwares for real-life applications. However, recent studies have shown deep learning architectures are highly vulnerable to attacks and exploitation via input perturbations. In this works, we investigate the effects of these attacks on the outputs of each layer of deep architectures and on theirs performance in term of classification accuracy measures. The results show that without defense mechanism, even simple attacks devastated deep architectures' outputs in every layer and theirs classification performance. We then propose multiple defense mechanisms in order to protect deep architectures and to make them more robust to input perturbation attacks.**

## I. Introduction

Recent advances in Deep Neural Networks (DNNs) have inspired mass applications of these deep computational architectures on multiple fields. The applicable fields include the classical ones such as Natural Language Processing, Speech Processing, Image Processing, and Computer Vision. Furthermore, developers have deployed DNNs in "hybrid" fields such as financial technology and recently, Internet of Things (IoTs). The applications of DNNs on various software and/or information systems, especially on embedded devices, requires us to consider them as possible targets in the "cyber kill chain" or "intrusion kill chain" or stages of cyber-attacks.

The danger of attacks on CNN is especially true when many deployed DNNs often re-use pre-trained networks with known architectures and published pre-trained weights. This creates a security situation called white-box attack. In a white-box attack, the attackers know exactly the victim's DNNs architectures, weights and training data. With these critical information, the attackers have very high chance of tweaking the input provided to the DNNs to acquired a desired output. For example, falsifying an authenticated face or other biometrics in order to get into a protected area.

On the other hand, deep learning, especially when following transfer learning approach, also leads to a reverse situation called black-box attack. In a black-box attack, the attackers use adversarial inputs obtained from attacking known DNNs (i.e. known architectures and known weights) to attack another *unknown or hidden* DNN model. This is probable because different DNN models could recognize the same input patterns and therefore, one could confuse them with the same adversarial inputs.

One could see that in both situation, defense against white-box attacks is therefore the first line of defense for DNNs. To this end, our contributions in this paper are

- Investigating the effects of attacks on the outputs of layers in a neural networks, especially the change in ranks and orders of the probability or logit output layer.
- Proposing a defense mechanism based on robust loss functions and empirically evaluating the effects of these loss functions against common attacks on the DNNs.

The paper is organized as followings: In Section II, we will explain several concepts in adversarial attack on neural networks and defense mechanisms for them. In Section III, we demonstrate the significant effects of simple and sophisticated attacks on a convolutional neural network and the need to defense against them. In Section IV, we propose a defense mechanism using established loss functions. We present experiment results in Section V and concluding remarks in Section VI.

## II. Background

### A. Adversarial examples

An adversarial example, was first discovered in [1], is an input which has been slightly modified to confuse a neural network to mis-classify it. Formally, a neural network is a function $F(x) = y$ that maps an input image $x \in R^n$ to a label $y \in \{1 \dots k\}$. This network classifies a *natural input image* $x$ to the target label $y = t$. The same network mis-classifies an *adversarial example* $x' = x + \delta$ to a different label $F(x') = t' \neq t$. The small perturbation, $\delta$, which is added to natural

978-1-5386-6113-0/18/$31.00 ©2018 IEEE

input image $x$, is normally bounded in $l_p$ norm in order to make the adversarial example be imperceptible for human to differentiate. A positive threshold $\epsilon$ is chosen such that $\|\delta\|_p < \epsilon$.

### B. Adversarial attacks

In order to attack a network, attackers often inspect the changes in output layers with respect to the changes in the input. Therefore, most adversarial attacks rely on first-order (gradient) information of a loss function. Formally, given the loss function, $L(x, y; w)$, of the model with respect to a pair of input $x$ and target label $y$ (while $w$ is the network weights), the adversary will use gradient, $\frac{\partial L}{\partial x}$, of this loss function to find an adversarial input that confuses the neural network. In the followings, we explains a few widely used attacking mechanisms in literature.

*1) Adversarial examples as solutions to an optimization problem:* Given a natural input $x \in [0, 1]^n$, one could pose the problem of finding a closest adversarial example $x'$ under $\ell_2$ distance as the following optimization

$$x' = \arg\min_{x'} \|x' - x\|_2^2$$

such that $F(x') = t' \neq t = F(x)$ and $x' \in [0, 1]^n$. A solution $x'$ to the above optimization is guaranteed to be mis-classified by the model $F$ while being as close as possible to the natural input $x$. As a result, the adversarial example is imperceptible for human to differentiate.

This optimization problem can be very difficult to solve, so the authors in [1] proposed to solve the following relaxed problem instead:

$$x' = \arg\min_{x'} c \cdot \|x' - x\|_2^2 + L(x', t') \qquad (1)$$

such that $x' \in [0, 1]^n$. This formulation allows one to employ available automatic gradient computation toolbox and training algorithms to optimize the objective function. In the relaxed formulation, the loss function $L$ is normally cross-entropy loss function. To yield an adversarial example of minimum distance, one solves optimization problem repeatedly with multiple and increasing values of $\ell_2$ distance penalization parameter $c$.

*2) CW method:* This method proposed in [2] aims to compute good approximations of Eq. (1) while keeping the computational cost of perturbing examples low. The authors cast the formulation into a more efficient optimization problem, which allows them to craft effective adversarial samples with low distortion. They define three similar targeted attacks, based on different distortion measures: $\ell_2$ , $\ell_0$ and $\ell_\infty$ respectively. However, in our experiments and in practice, even these attacks are computationally expensive.

*3) Fast gradient sign (FGS):* The method proposed in [3] finds an adversarial example that closes to natural image under $\ell_\infty$ norm and aims to find it approximately and quickly instead of finding the optimal solution. It finds an adversarial example by computing a very "crude" gradient descent step:

$$x' = x - \epsilon \cdot \text{sign}(\nabla_x L(x, t')).$$

Intuitively, the method looks into the gradient of the loss function to find the directions that minimize the loss function when classifying input $x$ as label $t'$. Then it either add to or subtract from the coordinates of the natural input a small amount, $\epsilon$, along the gradient sign direction to get $x'$ that will more likely be mis-classified as $t'$.

*4) Projected gradient descent (PGD):* This method is a refinement of FGS method where instead of taking a single step of size $\epsilon$ in the direction of gradient-sign, multiple smaller steps $\alpha$ are taken, and the result is clipped by the same $\epsilon$. Specifically, one initializes the algorithm by setting

$$x'_0 = x,$$

then on each iteration, compute

$$x'_i = \text{clip}_{x,\epsilon}(x'_{i-1} - \alpha \cdot \text{sign}(\nabla L(x'_{i-1}, t'))).$$

Iterative Projected Gradient Sign was found to produce superior results to Fast Gradient Sign. In [4], it is found that instead of starting at $x'_0 = x$, the natural input, one could add random perturbation of size $\epsilon$ to the natural input then performs iterative Projected Gradient Sign for a few iterations. Then selecting adversarial example that has minimum loss will lead to optimal adversarial example.

### C. Defense mechanisms

*1) Adversarial training:* The main idea in adversarial training is to make the model more robust to adversarial examples by introducing adversarial examples at training. The disadvantage of this approach is that it takes more computations to generate adversarial examples. FGS method can be used to quickly generate the adversarial examples, but the adversarial examples it generated are not optimal so it will fail when we use more complicated attacks. In [4], the authors showed that, iterative Projected Gradient Sign method with random restart can find the optimal adversarial examples and training on these adversarial examples makes the model robust against wide range of first-order attack.

*2) Distillation:* Distillation is a method described in [5] to transfer knowledge from a large model to a smaller model. [6] applied this method as a defense mechanism against adversary by hiding the gradient between the pre-softmax layer (logits) and softmax output. However, the

authors in [2] showed that it is easy to bypass the defense by using more complex attacks.

## III. Effect of attacks on current networks

In this section, we investigate the distortions of deep neural networks caused by adversarial examples. We use adversarial examples generated by random start PGD attack to be the universal adversary [4]. After that we apply the defense method, which is kind of adversarial training, proposed by the authors in [4] to see how it mitigates the effect of adversarial attack.

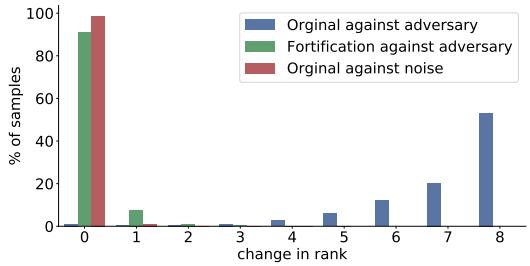


Fig. 1. The change in rank of predicted output on natural test set to modified test set.

### A. Attacking an MNIST model

TABLE I
MNIST network architecture

| Layer | # of units | Input shape | Output shape |
|---|---|---|---|
| Conv1 | 32 (5 x 5) | 28 x 28 x 1 | 28 x 28 x 32 |
| Pool1 | | 28 x 28 x 32 | 14 x 14 x 32 |
| Conv2 | 64 (5 x 5) | 14 x 14 x 32 | 14 x 14 x 64 |
| Pool2 | | 14 x 14 x 64 | 7 x 7 x 64 |
| FC1 | 1024 | 3136 | 1024 |
| FC2 | 10 | 1024 | 10 |

We study the effect of adversarial attack on a convolutional neural network used to classify hand written digits from MNIST dataset. This network consists of convolutional layers, max pooling layers and fully connected layers and uses ReLU as activation function. It accepts an grayscale image of size $28 \times 28$ and predicts the number in the input image. The architecture of this network is described in Table I. We train the model on MNIST training set using cross entropy softmax loss and Adam optimizer. This model achieves the accuracy of $99.13\%$ on MNIST test set. So the performance of this model is very high when dealing with natural input image.

Now we find the adversarial version of MNIST test set by applying random start PGD attack. We iterates $k = 100$ steps with size of each step $\alpha = 0.01$ to generate adversarial examples with $l_\infty$ norm bound $\epsilon = 0.3$. The accuracy of this model on adversarial test set is $0.27\%$. These adversarial examples are only slightly different from the natural examples but causes our model almost misclassifies all examples.

In Figure 1 we plot the change in rank of predicted output on natural test set to adversarial test set. We denote $y_i$ is the output of natural test sample $i$-th and $y'_i$ is the output of adversarial test sample $i$-th. These output are vectors in $10$ dimensions, each dimension index represents a number from $0$ to $9$. $rank(j, y)$ is a function that outputs the number of dimensions in output $y$ has higher value than dimension $j$, so if $rank(j, y) = 0$ then the number that the model predicts is $j$. First we find $j$ in the output of natural test sample that $rank(j, y_i) = 0$ then we find the rank of dimension $j$ in the output of adversarial test sample $r = rank(j, y'_i)$. We report $r$ as the change in rank of predicted output on natural test sample to adversarial test sample. The 0-change in rank means that the model classifies adversarial examples the same as natural examples. The 9-change in rank means that the model sees adversarial examples least likely to be the same as natural examples. As plotted, the change is almost 8. Therefore, adversarial examples do not only fool the model, they do it in the most intensive way.

To further investigate we plot the change in output of each layer on natural test set to adversarial test set. Let $x_{il}$ is the input goes into layer $l$-th of the natural test sample $i$-th, and $y_{il}$ is the output of that input. $x'_{il}$ and $y'_{il}$ is the input and output of layer $l$-th of the corresponding adversarial test sample $i$-th. We expect that if natural sample and adversarial sample are slightly different then each layer of the model will produce similar output, thus $y_{il}$ and $y'_{il}$ will be close in some measure. We choose $l_2$ norm to measure the similarity. Because each layer has different output dimension, we divide $l_2$ norm of the difference to $l_2$ norm of natural input $x_{il}$ so the difference between output of each layer will be proportional to natural input to that layer. Formally, with each layer we report $\frac{\|y'_{il} - y_{il}\|_2}{\|x_{il}\|_2}$.

The result is in Figure 2. We can see that the amount of change in each layer, which is at least half of natural input norm, is huge. The changes are accommodated through each layer, and make the final layer output very different to the natural samples.

Now we consider fetching noise samples into the model to see how each layer operates different to natural samples. To produce the noise samples, we randomly perturb all dimensions of the natural samples with the amount of perturbation never passes $\epsilon = 0.3$. Thus, all noise samples are close to natural samples in term of $l_\infty$ norm bound likes adversarial samples. The change of each layer when fetching noise samples is almost from $0.2$ to $0.4$ that is twice smaller than adversarial samples. That means the model is more resist to small random perturbation in input but not adversarial input. This result is consistent with the work in [3].

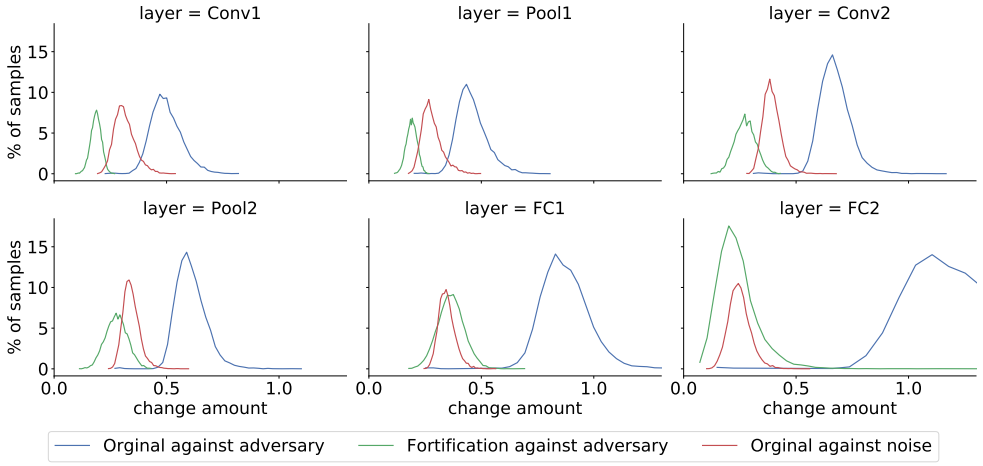


Fig. 2. The amount of change on each layer from natural test set to modified test set.

### B. *Effect of defense*

In the previous section, we can see that adversarial attack sabotage a neural network. Now we see how can we defense against that attack by training the network with adversarial examples in the framework proposed by [4].

The idea of this framework is that it first finds the most severe adversarial examples then using these adversarial examples to train the network so the network will be resistant to other adversarial attacks. To find the most severe adversarial input, we use PGD multiple times on natural input, each starts at a random point close to natural input then we pick the adversarial input that makes the network produce the highest loss. In practice, because we usually train a neural network on mini batch with multiple epochs, we don't necessarily find the most severe adversarial examples for each mini batch. Instead, we just use random PGD adversarial examples to train each mini batch. That saves us significant computing and produces the same result. In this experience, to generate PDG adversarial examples we set number of steps $k = 40$ and each step has size $\alpha = 0.01$, maximum perturbation in term $l_\infty$ norm bound $\epsilon = 0.3$.

To report the result we generate adversarial test set same as the section III-A. The accuracy is acceptable high (88.98%), so these adversarial examples cannot totally damage the as before. In addition, the accuracy performs on natural test set only slightly drop (97.73%). Hence adversarial training make the network still performs well on natural input but strongly resistant to adversarial examples.

Again, in Figure 2 we plot the change in output of each layer. We see that the amount of change in each layer is tiny. That means the network trained on this framework is robust against adversarial attacks. A small change amount in the input only produces a small change amount in output. To sum up, the adversarial training framework provides a decent way to make neural networks robust against adversarial attacks. In the following section, we will propose an improvement to this framework.

## IV. Change Loss Function in Adversarial Training Framework

[4] said that the adversarial training framework above works very well because it can find a nearly optimal solution to the minimax equation. The authors also stated that the capacity of the network to train on strong adversarial examples is important. Small capacity networks cannot learn anything meaningful from these strong adversarial examples. This suggests that the learning ability of neural networks take a crucial part toward a fully robust network against adversarial examples.

The goal is to find a network that can learn underlying concept of the object from the strong adversarial examples. At least, the network somehow learns to distinguish these strong adversarial examples. We hypothesize that exist a network architecture that is very robust against adversarial examples and there is a loss function that helps neural networks recognize adversarial examples as good as natural examples. There is a work that attempt to change the architecture of the network [7]. They proposed a new activation BReLU that is a bounded version of ReLU activation and helps the network not change much when input is adversarial.

In this paper, we do experiments to show that loss functions can help the networks learn better from these strong adversarial examples provided by the framework. And as we know, no one investigates about loss function as a defense mechanism against adversarial attack in literature. We will describe the experiments to study the

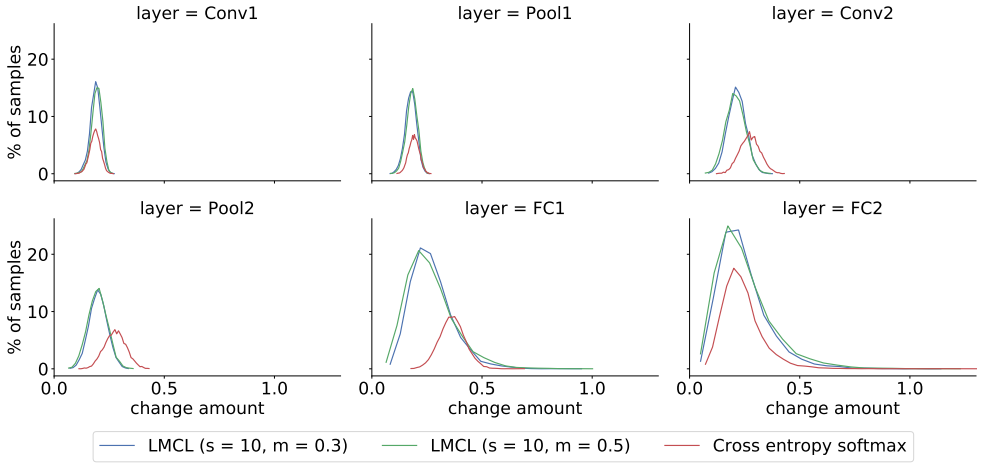


Fig. 3. LMCL loss vs. cross-entropy loss: Change amount layers' output.

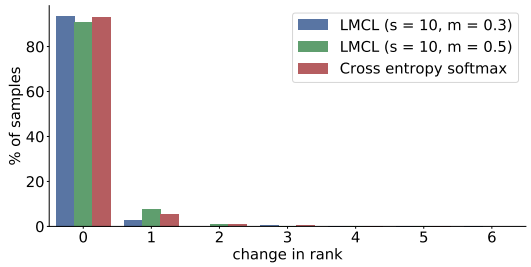


Fig. 4. LMCL loss vs. cross-entropy loss: Change in rank of target labels in models' output.

effect of some loss functions as a defense mechanism against adversarial attack in the following.

TABLE II
DEFENSE CAPABILITY OF DIFFERENT LOSS FUNCTIONS

| Loss Function | Adv. Acc. (%) | Nat. Acc. (%) |
|---|---|---|
| Hinge Loss | 64.97 | 91.04 |
| Cross Entropy Softmax Loss | 88.98 | 97.73 |
| LMCL (s = 10, m = 0.1) | 89.93 | 98.28 |
| LMCL (s = 10, m = 0.3) | 91.95 | 98.04 |
| LMCL (s = 10, m = 0.5) | 90.53 | 97.28 |
| LMCL (s = 10, m = 1.11) | 90.25 | 97.66 |

### A. *Loss functions*

Normally, we use cross entropy softmax loss to train multi class model. The cross entropy softmax loss for i-th data sample is written as following:

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

where $y_i$ is the correct class of i-th sample, $x_i$ is the input to the last layer, $f_j$ is the logit of class $j$, $f_j = W_j^T x_{ij}$. This loss wants the predicted distribution to have all of its mass on the correct class or probability of the correct class is as close to 1 as possible.

Another commonly used loss is hinge loss. It's usually used in multi-class support vector machine (SVM). The hinge loss is set up so that the SVM wants the correct class for each image to have a higher score than the incorrect classes by some fixed margin $\Delta$. In practice, we can safely set $\Delta = 1$ because the weights of model can shrink or stretch arbitrarily. The formula of the hinge loss is written as following:

$$L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1)$$

Adversarial examples are close to natural examples, but still have different predicted class. We think that if the model can distinguish each class very well then we can ease the adversarial problem. In other words, the decision boundary has a large margin to distinguish between classes.

There is another loss that focuses on large margin decision boundary. It's called large margin cosine loss (LMCL) proposed by [8]. The authors view cross entropy softmax loss from cosine perspective. They write $f_j$ in formula of the cross entropy loss as following:

$$f_j = W_j^T x_i = \|W_j\| \, \|x_i\| \cos\theta_j$$

Look at this formula, we can see that both norm and angle of vectors are contributed to predicted probability. If we fix both norm of vectors then our models will focus on learning the difference in the angle of them. The authors add a margin $m$ to make the model learning more robust:

$$L_i = -\log \frac{e^{s(\cos\theta_{yi} - m)}}{e^{s(\cos\theta_{yi} - m)} + \sum_{j \neq y_i} e^{s\cos\theta_j}}$$

subject to

$$W = \frac{W^*}{\|W^*\|}, x = \frac{x^*}{\|x^*\|}, \cos\theta_j = W_j^T x_i$$

TABLE III
THE TRANSFERABILITY OF ADVERSARIAL EXAMPLES GENERATED BY MULTIPLE MODELS

| Attack/Defense | Hinge loss | Softmax loss | LMCL (s = 10, m = 0.3) | Attack Average |
|---|---|---|---|---|
| Hinge loss | 64.97 | 96.26 | 95.9 | 85.71 |
| Softmax loss | 85.53 | 88.98 | 94.74 | 89.75 |
| LMCL (s = 10, m = 0.3) | 85.45 | 95.53 | 91.95 | 90.98 |
| Defense Average | 78.65 | 93.59 | 94.20 | |

We train multiple models using the same architecture as in Table I but using different loss functions: cross entropy softmax loss, hinge loss and large margin cosine loss. We train the model using the adversarial training framework with hyper parameter: $k = 40, \alpha = 0.01, \epsilon = 0.3$. We fix random seed, iteration steps (100000 steps) and use Adam optimizer. After training all models, we attack the models using random start PGD attack with hyper parameters: $k = 100, \alpha = 0.01, \epsilon = 0.3$. Then we report the accuracy of each model on corresponding adversarial test set and natural test set.

## V. EXPERIMENT RESULTS

To see that using different loss functions whether helps or not, we train multiple models with same architecture using different loss functions as listed in previous section. With large margin cosine loss, there are two hyperparameters $s$ and $m$ need to be set. Where $s$ is the fixed norm of input vectors going to loss layer and $m$ is the margin between each class in cosine space. We arbitrarily choose $s = 10$ and vary $m$ from $0.1$ to $1.11$ (theoretical upper bound that calculated by the formula in [8]). The results are reported in Table II.

The hinge loss has the worst accuracy on both adversarial test set and natural test set. Perhaps the hinge loss is useful to punish misclassifications (thus determine margin) but it does not help at probability estimation. In practice, people also rarely use hinge loss in deep learning.

All models trained with LMCL have better accuracy. That means LMCL really makes models better at distinguishing between classes and reduces the effect of adversarial attacks. To look closer, in Figure 3 we plot the change in amount of each layer of model trained with LMCL compare with original cross entropy softmax in Section III. As we can see the amounts of change are smaller in all layers, that confirms the efficiency of LMCL. We also plot the change in rank of predicted output in Figure 4.

Now we examine the transferability of adversarial examples generated from these models. We use adversarial examples generated from one model to attack another model then report the classification accuracy. The purpose of this experiment is showing that if the model is robust against adversarial examples then PGD attack will be struggle to find strong adversarial examples. Thus it will not effective to use these adversarial examples to attack other models. In addition, the robust model will show high resistance to adversarial examples generated by other models. The results are reported in Table III. The model trained with LMCL has the highest average accuracy when using as attacker that means adversarial examples generated by this model are easy to other models. When using this model as defender, it also has the highest average accuracy, that means it is difficult to other models to generate severe adversarial examples to attack this model.

## VI. CONCLUSION

DNNs have been demonstrated their superior performance on many tasks, but they are vulnerable to adversarial examples. In this paper, we invest the effect of adversarial attack on DNNs and review the state-of-the-art framework that makes DNNs robust. We propose to use large margin cosine loss instead of cross entropy softmax loss in the framework. The experiment results show that our proposal really improve the current framework.

## ACKNOWLEDGEMENT

This work has been supported by VNU University of Engineering and Technology under project number CN17.09.

## REFERENCES

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013. [Online]. Available: http://arxiv.org/abs/1312.6199

[2] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," *CoRR*, vol. abs/1608.04644, 2016. [Online]. Available: http://arxiv.org/abs/1608.04644

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *ArXiv e-prints*, Dec. 2014.

[4] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks," *ArXiv e-prints*, Jun. 2017.

[5] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *ArXiv e-prints*, Mar. 2015.

[6] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks," *ArXiv e-prints*, Nov. 2015.

[7] V. Zantedeschi, M.-I. Nicolae, and A. Rawat, "Efficient Defenses Against Adversarial Attacks," *ArXiv e-prints*, Jul. 2017.

[8] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu, "CosFace: Large Margin Cosine Loss for Deep Face Recognition," *ArXiv e-prints*, Jan. 2018.