DeepDefense: Training Deep Neural Networks with Improved Robustness

Ziang Yan[1*] Yiwen Guo[2*] Changshui Zhang[1]
[1]Department of Automation, Tsinghua University
State Key Laboratory of Intelligent Technology and Systems
Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing, China
[2] Intel Labs, China
yza15@mails.tsinghua.edu.cn yiwen.guo@intel.com zcs@mail.tsinghua.edu.cn

Abstract

Despite the efficacy on a variety of computer vision tasks, deep neural networks (DNNs) are vulnerable to adversarial attacks, limiting their applications in security-critical systems. Recent works have shown the possibility of generating imperceptibly perturbed image inputs (a.k.a., adversarial examples) to fool well-trained DNN models into making arbitrary predictions. To address this problem, we propose a training recipe named DeepDefense. Our core idea is to integrate an adversarial perturbation-based regularizer into the classification objective, such that the obtained models learn to resist potential attacks, directly and precisely. The whole optimization problem is solved just like training a recursive network. Experimental results demonstrate that our method outperforms the state-of-the-arts by large margins on various datasets (including MNIST, CIFAR-10 and ImageNet) and different DNN architectures. Code and models for reproducing our results will be made publicly available.

1. Introduction

Although deep neural networks (DNNs) have advanced the state-of-the-art of many challenging computer vision tasks, they are vulnerable to *adversarial examples* [32] (*i.e.*, generated images which are perceptually similar to the real ones and intentionally formed to fool learning models).

A general way of synthesizing the adversarial examples is to apply worst-case perturbations to real images [32, 7, 26]. With proper strategies, the required perturbations for fooling a DNN model can be 1000× smaller in magnitude when compared with the real images, making them imperceptible to human beings. It has been reported that even the state-of-the-art DNN solutions have been fooled to misclassify such examples with high confidence [19]. Worse, the adversarial perturbation can transfer across different images and network architectures [25]. Such transferability also allows black-box attacks, which mean the adversary may succeed without having any knowledge about the model architecture or parameters [28].

Though intriguing, such property of DNNs can lead to potential issues in real-world applications (*e.g.* self-driving cars and paying with your face systems). Unlike the instability against random noise, which is theoretically and practically guaranteed to be less critical [6, 32], the vulnerability to adversarial perturbations is still severe in deep learning. Multiple attempts have been made to analyze and explain it so far [32, 7, 5, 12]. For example, Goodfellow *et al*. [7] argue the main reason why DNNs are vulnerable is their linear nature instead of nonlinearity and overfitting. Based on the explanation, they design an efficient linear perturbation and further propose to combine it with adversarial training [32] for better regularization. Recently, Cisse *et al*. [5] investigate the Lipschitz constant of DNN-based classifiers and propose Parseval training to control it for improved robustness. However, similar with some previous regularization-based methods [8], Parseval training requires some approximations to its theoretically optimal constraint, making it less effective to resist very strong attacks.

In this paper, we introduce *DeepDefense*, an adversarial regularization method to train DNNs with improved robustness. Unlike many existing and contemporaneous methods which make approximations and optimize untight bounds, we precisely integrate a perturbation-based regularizer into the classification objective. This endows the obtained DNN models with an ability of directly learning from adversarial attacks and further resisting them, in a principled way. Specifically, we penalize the norm of adversarial perturbations, by encouraging relatively large values for the correctly classified samples and possibly small values for those misclassified ones. As a regularizer, it is jointly optimized with the original learning objective and the whole problem is efficiently solved through being considered as training

*The first two authors contributed equally to this work.

a recursive-flavoured network. Extensive experiments on MNIST, CIFAR-10 and ImageNet show that our method significantly improves the robustness of different DNNs under advanced adversarial attacks, in the meanwhile **no accuracy degradation is observed**.

The remainder of this paper is structured as follows. First, we briefly introduce and discuss representative methods for conducting adversarial attacks and defenses in Section 2. Then we elaborate on the motivation and basic ideas of our method in Section 3. Section 4 provides implementation details of our method and experimentally compares it with the state-of-the-arts, and finally, Section 5 draws the conclusions.

2. Related Work

Adversarial Attacks. Starting from a common objective, many methods have been proposed. Szegedy *et al*. [32] propose to generate adversarial perturbations by minimizing a vector norm using box-constrained L-BFGS optimization. For better efficiency, Goodfellow *et al*. [7] develop fast gradient sign (FGS) attack, which chooses the sign of gradient as the direction of perturbation since it is "approximately" optimal under the ℓ_∞ constraint. Later, Kurakin *et al*. [18] present a iterative version of FGS attack by applying gradient sign attack multiple times with a small step size, and clipping pixel values on internal results. Recently, Moosavi-Dezfooli *et al*. [26] propose DeepFool as an accurate iterative method. At each iteration, it linearizes the network and seeks the smallest perturbation to transform current images towards the linearized decision boundary. Some more detailed explanations of DeepFool can be found in Section 3.1.

Based on the above methods, input- and network- agnostic adversarial examples can also be generated. For example, Moosavi-Dezfooli *et al*. [25] have shown the existence of a universal adversarial perturbation that leads a DNN to misclassify many different input images with high probability. Furthermore, this universal perturbation also transfers well across different network architectures. Such transferability allows black-box attacks—the attacks without knowing about the network architecture and model parameters. Recently, Papernot *et al*. [28] demonstrate a black-box attack against models hosted by Amazon and Google, making them misclassify examples at rates of 96.19% and 88.94%, respectively, indicating that safety is not guaranteed even on private systems.

Some recent works also show that adversarial examples exist in semantic segmentation, object detection [34] and reinforcement learning tasks [14]. In addition, even if the adversary is unable to feed data directly into the DNN, the system can also be fooled by adversarial examples in the physical world [18].

Defenses. Resisting adversarial attacks is challenging. It has been empirically studied that conventional regularization strategies such as dropout, weight decay and distorting training data (with random noise) do not really solve the problem [7]. Fine-tuning networks using adversarial examples, namely adversarial training [32], is a simple yet effective approach to perform defense and relieve the problem [7, 19], for which the examples can be generated either online [7] or offline [26]. Adversarial training works well on small datasets such as MNIST and CIFAR. Nevertheless, as Kurakin *et al*. [19] have reported, it may result in a decreased test-set accuracy on large-scale datasets like ImageNet. Another effective work is proposed by Papernot *et al*. [29], which exploit distillation to resist attacks, but it also slightly degrades the test-set accuracy. Alemi *et al*. [3] present an information theoretic method which helps on improving the resistance to adversarial attacks too.

An alternative way of defending such attacks is to train a detector to detect and reject adversarial examples. Metzen *et al*. [23] utilize a binary classifier which takes intermediate representations as input for detection, and Lu *et al*. [22] propose to invoke an RBF-SVM operating on discrete codes from late stage ReLUs. However, it is still possible to perform attacks on the joint system if an adversary has access to the parameters of such a detector, since the detector itself is also a classifier and can be similarly fooled. Furthermore, it is still in doubt whether the adversarial examples are intrinsically different with the clean images [4].

Several regularization-based methods have also been proposed. For example, Gu and Rigazio [8] propose to penalize the Frobenius norm of the Jacobian matrix in a layer-wise fashion. Recently, Cisse *et al*. [5] theoretically show the sensitivity to adversarial examples can be controlled by the Lipschitz constant of the network and propose Parseval training. However, these methods usually require rough approximations, making them less effective to defend very strong and advanced adversarial attacks.

As a regularization-based method, our DeepDefense is orthogonal to the adversarial training, defense distillation and detecting then rejecting methods. It differs from previous regularization-based methods (*e.g*. [8, 5, 12]) in a way that we endow DNNs the ability to precisely resist adversarial examples by directly learning from them.

3. Our DeepDefense Method

Many previous methods regularize the learning process of DNNs approximately, which may lead to a degraded accuracy on test sets or unsatisfactory robustness to advanced adversarial examples. We reckon that it could be more beneficial to directly incorporate attack modules into the learning objective and precisely learn to resist them. In this section, we first briefly analyze a representative gradient-based attack and then introduce our solution to learn from it.

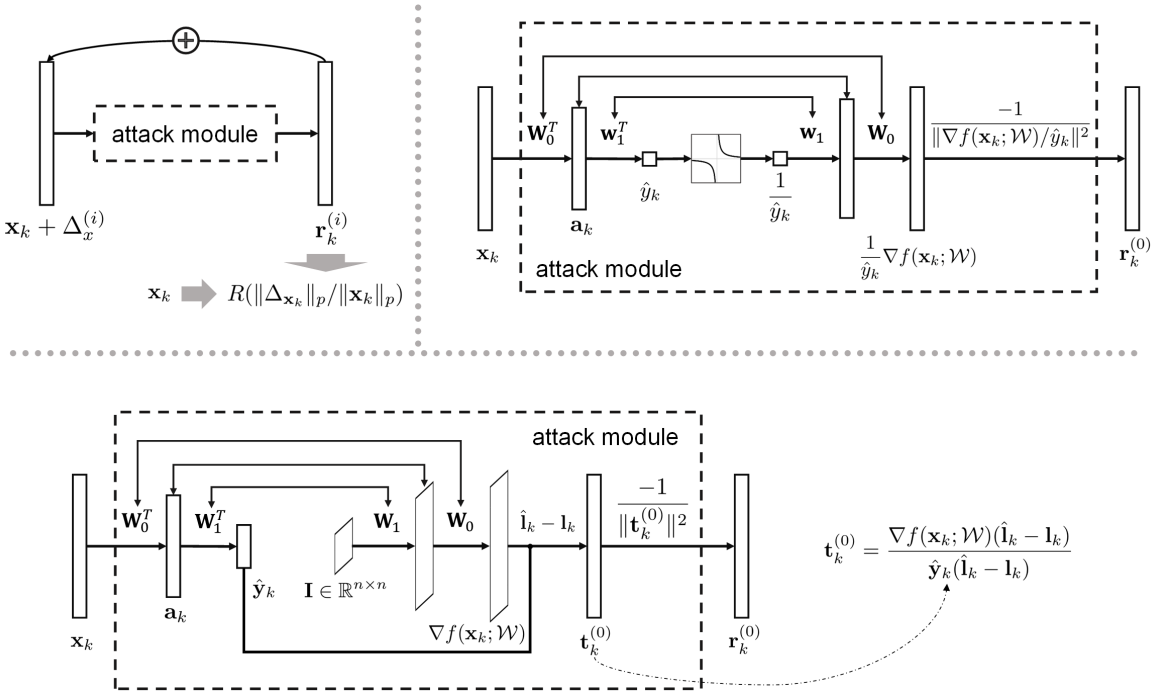

Figure 1: Top left: The recursive-flavoured network which takes a reshaped image $\mathbf{x}_k$ as input and sequentially compute each $\mathbf{r}_k^{(i)}$ (for $0 \leq i < u$) by using a pre-designed attack module. Top right: an example for generating $\mathbf{r}_k^{(0)}$ with the incorporated attack module, in which the three elbow double-arrow connectors, from top to bottom, indicate weight-sharing between the first and fourth fully-connected layers, index-sharing between the two ReLU activation layers, and weight-sharing between the second and third fully-connected layers, respectively. Bottom: the attack module for n-class ($n \geq 2$) scenarios, in which $\hat{\mathbf{y}}_k \in \mathbb{R}^n$ indicates the network output before softmax, $\mathbf{I} \in \mathbb{R}^{n\times n}$ is the identity matrix, and $\hat{\mathbf{l}}_k \in \{0,1\}^n$ and $\mathbf{l}_k \in \{0,1\}^n$ indicate the one-hot encoding of current prediction label and a chosen label to fool in the first iteration, respectively.

3.1. Generate Adversarial Examples

It is popular to conduct attacks based on iterative linearization. Let us take DeepFool [26] as an example, since it is regarded as one of the most effective methods by far [2]. Mathematically, starting from a binary classifier $f : \mathbb{R}^m \to \mathbb{R}$ which makes prediction (to the class label) based on the sign of its output, DeepFool generates the adversarial perturbation $\Delta_{\mathbf{x}}$ for an arbitrary input vector $\mathbf{x} \in \mathbb{R}^m$ in a heuristic way. Concretely, $\Delta_{\mathbf{x}} = \mathbf{r}^{(0)} + ... + \mathbf{r}^{(u-1)}$, in which the i-th ($0 \leq i < u$) addend $\mathbf{r}^{(i)}$ is obtained by taking advantage of the Taylor's theorem and solving:

$$\min_{\mathbf{r}} \|\mathbf{r}\|_2 \quad \text{s.t. } f(\mathbf{x} + \Delta_{\mathbf{x}}^{(i)}) + \nabla f(\mathbf{x} + \Delta_{\mathbf{x}}^{(i)})^T \mathbf{r} = 0, \quad (1)$$

in which $\Delta_{\mathbf{x}}^{(i)} := \sum_{j=0}^{i-1} \mathbf{r}^{(j)}$, function ∇f denotes the gradient of f w.r.t. its input, and operator $\|\cdot\|_2$ denotes the Euclidean norm. Obviously, Equation (1) has a closed-form solution as:

$$\mathbf{r}^{(i)} = -\frac{f(\mathbf{x} + \Delta_{\mathbf{x}}^{(i)})}{\|\nabla f(\mathbf{x} + \Delta_{\mathbf{x}}^{(i)})\|^2} \nabla f(\mathbf{x} + \Delta_x^{(i)}). \quad (2)$$

By sequentially calculating all the $\mathbf{r}^{(i)}$s with (2), DeepFool employs a faithful approximation to the $\Delta_{\mathbf{x}}$ of minimal norm. In general, the approximation algorithm converges in a reasonably small number of iterations even when f is a non-linear function represented by a very deep neural network, making it both effective and efficient in practical usage. The for-loop for calculating $\mathbf{r}^{(i)}$s ends in advance if the attack goal $\text{sgn}(f(\mathbf{x} + \Delta_{\mathbf{x}}^{(i)})) \neq \text{sgn}(f(\mathbf{x}))$ is already reached at any iteration $i < u - 1$. Similarly, such strategy also works for the adversarial attacks to multi-class classifiers, which only additionally require a specified target label in each iteration of the algorithm.

3.2. Perturbation-based Regularization

Our target is to improve the robustness of off-the-shell networks without modifying their architectures, hence giving a $\|\Delta_{\mathbf{x}}\|_p$-based ($p \in [1, \infty)$) regularization to their original objective function seems to be a solution.

Considering the aforementioned attacks which utilize ∇f when generating the perturbation $\Delta_{\mathbf{x}}$ [32, 7, 26, 34], their strategy can be technically regarded as a function pa-

rameterized by the same set of learnable parameters as that of f. Therefore, it is possible that we jointly optimize the original network objective and a scaled $\|\Delta_{\mathbf{x}}\|_p$ as regularization for some chosen norm operator $\|\cdot\|_p$, provided $\|\Delta_{\mathbf{x}}\|_p$ is differentiable. Specifically, given a set of training samples $\{(\mathbf{x}_k, \mathbf{y}_k)\}$ and a parameterized function f, we may want to optimize:

$$\min_{\mathcal{W}} \sum_k L(\mathbf{y}_k, f(\mathbf{x}_k; \mathcal{W})) + \lambda \sum_k R\left(-\frac{\|\Delta_{\mathbf{x}_k}\|_p}{\|\mathbf{x}_k\|_p}\right), \quad (3)$$

in which the set $\mathcal{W}$ exhaustively collects learnable parameters of f, and $\|\mathbf{x}_k\|_p$ is a normalization factor for $\|\Delta_{\mathbf{x}_k}\|_p$, as adopted in Moosavi-Dezfooli *et al.*'s work [26]. As will be further detailed in Section 3.4, function R should treat incorrectly and correctly classified samples separately, and it should be monotonically increasing on the latter such that it gives preference to those fs resisting small $\|\Delta_{\mathbf{x}_k}\|_p/\|\mathbf{x}_k\|_p$ anyway. Regarding the DNN representations, $\mathcal{W}$ may comprise the weight and bias of network connections, means and variances of batch normalization layers [15], and slops of the parameterized ReLU layers [11].

3.3. Network-based Formulation

As previously discussed, we should re-formulate the adversarial perturbation as $\Delta_{\mathbf{x}_k} = g(\mathbf{x}_k; \mathcal{W})$, in which g need to be differentiable except for maybe certain points, so that problem (3) can be solved using stochastic gradient descent and following the chain rule. In order to make the computation more efficient and easily parallelized, an explicit formulation of g or its gradient w.r.t $\mathcal{W}$ is required. Here we accomplish this task by representing g as a "reverse" network to the original one. Taking a two-class multi-layer perceptron (MLP) as an example, we have $\mathcal{W} = \{\mathbf{W}_0, \mathbf{b}_0, \mathbf{w}_1, b_1\}$ and

$$f(\mathbf{x}_k; \mathcal{W}) = \mathbf{w}_1^T h(\mathbf{W}_0^T \mathbf{x}_k + \mathbf{b}_0) + b_1, \quad (4)$$

in which h denotes the non-linear activation function and we choose $h(\mathbf{W}_0^T \mathbf{x}_k + \mathbf{b}_0) := \max(\mathbf{W}_0^T \mathbf{x}_k + \mathbf{b}_0, \mathbf{0})$ (*i.e.* as the ReLU activation function) in this paper since it is commonly used. Let us further denote $\mathbf{a}_k := h(\mathbf{W}_0^T \mathbf{x}_k + \mathbf{b}_0)$ and $\hat{y}_k := f(\mathbf{x}_k; \mathcal{W})$, then we have

$$\nabla f(\mathbf{x}_k; \mathcal{W}) = \mathbf{W}_0(\mathbf{1}_{>0}(\mathbf{a}_k) \otimes \mathbf{w}_1), \quad (5)$$

in which $\otimes$ indicates the element-wise product of two matrices, and $\mathbf{1}_{>0}$ is an element-wise indicator function that compares the entries of its input with zero.

Here we still choose $\Delta_{\mathbf{x}_k}$ as the previously discussed DeepFool perturbation for simplicity of notations [1]. Based on Equation (2) and (5), we construct a recursive-flavoured regularizer network (as illustrated in the top left of Figure 1) to calculate $R(-\|\Delta_{\mathbf{x}_k}\|_p/\|\mathbf{x}_k\|_p)$. It takes image $\mathbf{x}_k$ as input and calculate each addend for $\Delta_{\mathbf{x}_k}$ by utilizing an incorporated multi-layer attack module (as illustrated in the top right of Figure 1). Apparently, the original three-layer MLP followed by a multiplicative inverse operator makes up the first half of the attack module and its "reverse" followed by a norm-based rescaling operator makes up the second half. It can be easily proved that the designed network is differentiable w.r.t. the elements of $\mathcal{W}$. As sketched in the bottom of Figure 1, such a network-based formulation can also be naturally generalized to regularize multi-class MLPs with more than one output neurons (*i.e.*, $\hat{\mathbf{y}}_k \in \mathbb{R}^n$, $\nabla f(\mathbf{x}_k; \mathcal{W}) \in \mathbb{R}^{m \times n}$ and $n > 1$).

Seeing that current winning DNNs are constructed as a stack of convolution, non-linear activation (*e.g.*, ReLU, parameterized ReLU and sigmoid), normalization (*e.g.*, local response normalization [17] and batch normalization), pooling and fully-connected layers, their ∇f functions, and thus g functions, should be differentiable almost everywhere. Consequently, feasible "reverse" layers can always be made available to these popular layer types. In addition to the above explored ones (*i.e.*, ReLU and fully-connected layers), we also have deconvolution layers [27] which are reverse to the convolution layers, and unpooling layers [36] which are reverse to the pooling layers, *etc*. Just note that some learning parameters and variables like filter banks and pooling indices should be shared between them.

3.4. Robustness and Accuracy

Problem (3) integrates an adversarial perturbation-based regularization into the classification objective, which should endow parameterized models with the ability of learning from adversarial attacks and resisting them for improved robustness. Additionally, it is also crucial not to degrade the inference accuracy of these models. Goodfellow *et al.* [7] have shown the possibility of fulfilling such expectation in a data augmentation manner. Here we explore more on our robust regularization to ensure it does not degrade test-set accuracies either.

Most attacks treat all the input samples equally [32, 7, 26, 19], regardless of whether or not their predictions match the ground-truth labels. It makes sense when we aim to fool the networks, but not when we leverage the attack module to supervise training. Specifically, we might expect a decrease in $\|\Delta_{\mathbf{x}_k}\|_p/\|\mathbf{x}_k\|_p$ from any misclassified sample $\mathbf{x}_k$, especially when the network is to be "fooled" to classify it as its ground-truth. This seems different with the objective as formulated in (3), which appears to enlarge the adversarial perturbations for all training samples.

Moreover, we found it difficult to seek reasonable tradeoffs between robustness and accuracy, if R is a linear function (*e.g.*, $R(z) = z$). In that case, the regularization term is dominated by some extremely "robust" samples, so the

[1] Note that our method also naturally applies to some other gradient-based adversarial attacks.

Dataset	Network	Method	Acc.	ρ_2	ρ_∞	Acc.@$0.2\epsilon_{\text{ref}}$	Acc.@$0.5\epsilon_{\text{ref}}$	Acc.@$1.0\epsilon_{\text{ref}}$
MNIST	MLP	Reference	98.31%	1.11×10^{-1}	5.40×10^{-2}	72.76%	29.08%	3.31%
		Par. Train [5]	98.32%	1.11×10^{-1}	5.78×10^{-2}	77.44%	28.95%	2.96%
		Adv. Train [26]	98.49%	1.62×10^{-1}	9.46×10^{-2}	87.70%	59.69%	22.55%
		DeepDefense	**98.65%**	$\mathbf{2.25\times10^{-1}}$	$\mathbf{1.53\times10^{-1}}$	**95.04%**	**88.93%**	**50.00%**
	LeNet	Reference	99.02%	2.05×10^{-1}	1.29×10^{-1}	90.95%	53.88%	19.75%
		Par. Train [5]	99.10%	2.03×10^{-1}	1.50×10^{-1}	91.68%	66.48%	19.64%
		Adv. Train [26]	99.18%	2.63×10^{-1}	2.05×10^{-1}	95.20%	74.82%	41.40%
		DeepDefense	**99.34%**	$\mathbf{2.84\times10^{-1}}$	$\mathbf{2.36\times10^{-1}}$	**96.51%**	**88.93%**	**50.00%**
CIFAR-10	ConvNet	Reference	79.74%	2.59×10^{-2}	5.27×10^{-3}	61.62%	37.84%	23.85%
		Par. Train [5]	80.48%	3.42×10^{-2}	8.02×10^{-3}	69.19%	50.43%	22.13%
		Adv. Train [26]	80.65%	3.05×10^{-2}	6.37×10^{-3}	65.16%	45.03%	35.53%
		DeepDefense	**81.70%**	$\mathbf{5.32\times10^{-2}}$	$\mathbf{1.58\times10^{-2}}$	**72.15%**	**59.02%**	**50.00%**
	NIN	Reference	89.64%	4.20×10^{-2}	1.05×10^{-2}	75.61%	49.22%	33.56%
		Par. Train [5]	88.20%	4.33×10^{-2}	1.07×10^{-2}	75.39%	49.75%	17.74%
		Adv. Train [26]	89.87%	5.25×10^{-2}	1.76×10^{-2}	78.87%	58.85%	45.90%
		DeepDefense	**89.96%**	$\mathbf{5.58\times10^{-2}}$	$\mathbf{2.15\times10^{-2}}$	**80.70%**	**70.73%**	**50.00%**
ImageNet	AlexNet	Reference	56.91%	2.98×10^{-3}	5.46×10^{-4}	54.62%	51.39%	46.05%
		DeepDefense	**57.11%**	$\mathbf{4.54\times10^{-3}}$	$\mathbf{8.70\times10^{-4}}$	**55.79%**	**53.50%**	**50.00%**
	ResNet-18	Reference	66.63%	1.90×10^{-3}	7.56×10^{-4}	60.02%	50.45%	37.66%
		DeepDefense	**66.81%**	$\mathbf{3.68\times10^{-3}}$	$\mathbf{1.46\times10^{-3}}$	**63.25%**	**58.08%**	**50.00%**

Table 1: Test set performance of different defense methods under adversarial attacks. Column 4: accuracies on clean test images. Column 5: ρ_2 values under the DeepFool attack. Column 6: ρ_∞ values under the Fast Gradient Sign (FGS) attack. Column 7-9: accuracies on the FGS perturbed images, and ϵ_{ref} is chosen as the smallest ϵ such that 50% perturbed images are misclassified by our regularized model (*i.e.*, different ϵ_{ref} values for different networks).

training samples with relatively small $\|\Delta_{\mathbf{x}_k}\|_p/\|\mathbf{x}_k\|_p$ are not fully optimized. This phenomenon can impose negative impact on the classification objective L and thus inference accuracy. In fact, for those samples which are already robust enough, enlarging $\|\Delta_{\mathbf{x}_k}\|_p/\|\mathbf{x}_k\|_p$ is not really necessary. It is appropriate to penalize more on the currently correctly classified samples with abnormally small $\|\Delta_{\mathbf{x}_k}\|_p/\|\mathbf{x}_k\|_p$ values than those with relatively large values (*i.e.*, the ones already been considered robust with f and $\Delta_{\mathbf{x}_k}$).

To this end, we rewrite the second term in the objective function of Problem (3) as

$$\lambda \sum_{k\in\mathcal{T}} R\left(-c\frac{\|\Delta_{\mathbf{x}_k}\|_p}{\|\mathbf{x}_k\|_p}\right) + \lambda \sum_{k\in\mathcal{F}} R\left(d\frac{\|\Delta_{\mathbf{x}_k}\|_p}{\|\mathbf{x}_k\|_p}\right), \quad (6)$$

in which $\mathcal{F}$ is the index set of misclassified training samples, $\mathcal{T}$ is its complement, $c, d > 0$ are two scaling factors that balance the importance of different samples, and R is chosen as the exponential function. With extremely small or large c, our method treats all the samples the same in $\mathcal{T}$, otherwise those with abnormally small $\|\Delta_{\mathbf{x}_k}\|_p/\|\mathbf{x}_k\|_p$ will be penalized more than the others.

4. Experimental Results

In this section, we evaluate the efficacy of our DeepDefense method on three different datasets: MNIST, CIFAR-10 and ImageNet [30]. We compare it with two state-of-the-art methods: adversarial training and Parseval training (also known as Parseval networks). Similar with some previous works [26, 3], we choose to fine-tune from pre-trained models instead of training from scratch. Fine-tuning hyper-parameters can be found in Table 2. All the experiments are implemented using PyTorch [1] on an NVIDIA GTX 1080 GPU. Our main results are summarized in Table 1, where the fourth column shows the inference accuracy of different models on clean test images, the fifth column compares the robustness of different models to DeepFool adversarial examples, and the subsequent columns compare the robustness to FGS adversarial examples. The evaluation metrics are carefully explained in Section 4.1. Some implementation details of the compared methods are shown as follows.

Dataset	Batch Size	#Epoch	Base Learning Rate
MNIST	100	5	5×10^{-4}
CIFAR-10	100	50	5×10^{-4}
ImageNet	256	10	1×10^{-4}

Table 2: Some hyper-parameters in the fine-tuning process.

DeepDefense. There are three hyper-parameters in our method: λ, c and d. As previously explained in Section 3.4, they balance the importance of model robustness and test-set accuracy. We uniformly set $\lambda = 15, c = 25, d = 5$ for

MNIST and CIFAR-10 main experiments, and $\lambda = 5, c = 500, d = 5$ for ImageNet experiments. The Euclidean norm is simply chosen for $\|\cdot\|_p$. Practical impact of varying these hyper-parameters will be discussed in Section 4.2.

Adversarial Training. There exists many different versions of adversarial training [32, 7, 26, 19, 24], partly because it can be combined with different attack methods. We choose the one introduced by Moosavi-Dezfooli *et al.* [26] here and try out to reach its optimal performance, which means using a fixed adversarial training set generated by DeepFool, and no clean images are involved in the fine-tuning process.

Parseval Training. Parseval Networks [5] improve the robustness of a DNN by carefully controlling its global Lipschitz constant. Practically, a projection update is performed after each stochastic gradient descent (SGD) iteration to ensure all weight matrices' Parseval tightness. Following the original paper, we uniformly sample 30% of columns to perform this update. We set the hyper-parameter $\beta = 0.0001$ for MNIST, and $\beta = 0.0003$ for CIFAR-10 after doing grid search.

4.1. Evaluation Metrics

This subsection explains the evaluation metrics adopted in our experiments. We conduct the famous DeepFool and FGS attacks on different DNN architectures and compare the robustness of obtained models using different defense methods. As suggested in the paper [26], we evaluate such robustness by calculating

$$\rho_2 := \frac{1}{|\mathcal{D}|} \sum_{k \in \mathcal{D}} \frac{\|\Delta_{\mathbf{x}_k}\|_2}{\|\mathbf{x}_k\|_2}, \tag{7}$$

in which $\mathcal{D}$ is the test set (for ImageNet we use its validation set), when the DeepFool method is used to conduct attacks. Similarly, when the FGS attack is adopted, the robustness can be evaluated by replacing the l_2 norm with an l_∞ norm as the FGS attack is usually considered as an l_∞ norm-based (or max-norm based) perturbation method, *i.e.*,

$$\rho_\infty := \frac{1}{|\mathcal{D}|} \sum_{k \in \mathcal{D}} \frac{\|\Delta_{\mathbf{x}_k}\|_\infty}{\|\mathbf{x}_k\|_\infty}. \tag{8}$$

Recall that in the FGS method,

$$\Delta_{\mathbf{x}_k} = \epsilon \text{sign}\left(\nabla_{\mathbf{x}_k} L(\mathbf{y}_k, f(\mathbf{x}_k; \mathcal{W}))\right), \tag{9}$$

in which L is the original classification objective as introduced in Section 3, and $\mathcal{W}$ is still the set of learning parameters here, we adjust ϵ so that 50% of the image samples are misclassified by well-trained models. Apparently, higher ρ_2 and ρ_∞ indicate better robustness to the DeepFool and FGS adversarial examples, respectively.

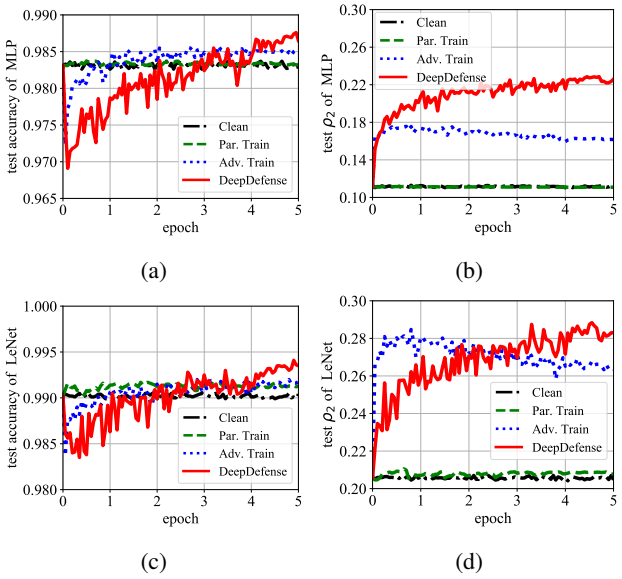

Figure 2: Convergence curves on MNIST: (a) test accuracy of MLP, (b) test ρ_2 of MLP, (c) test accuracy of LeNet, and (d) test ρ_2 of LeNet. "Clean" indicates fine-tuning on clean images. Best viewed in color.

It is also interesting to evaluate the accuracy on a perturbed $\mathcal{D}$ as an alternative metric for the FGS attack [8, 5]. Likewise, we first calculate the smallest ϵ such that 50% of the perturbed images are misclassified by our regularized models and denote it as ϵ_{ref}, then test the inference accuracies of those models produced by adversarial training and Parseval training at this level of perturbation (abbreviated as "Acc.@$1.0\epsilon_{\text{ref}}$ in Table 1"). Inference accuracies at lower levels of perturbations (a half and one fifth of ϵ_{ref}) are also reported.

4.2. Exploratory Experiments on MNIST

As a popular dataset for conducting adversarial attacks [32, 7, 26], MNIST is a reasonable choice for us to get started. It consists of 70,000 grayscale images, in which 60,000 of them are used for training and the remaining are used for test. We train a four-layer MLP and download a LeNet [20] structured CNN model [2] as references (see supplementary material for more details).

For fair comparisons, we use identical fine-tuning policies and hyper-parameters for different defense methods (as summarized in Table 2). We cut the learning rate by $2\times$ after four epochs of training because it can be beneficial for convergence. Other hyper-parameters like momentum and weight decay are kept as the same as training the reference

[2] https://github.com/LTS4/DeepFool/blob/master/MATLAB/resources/net.mat

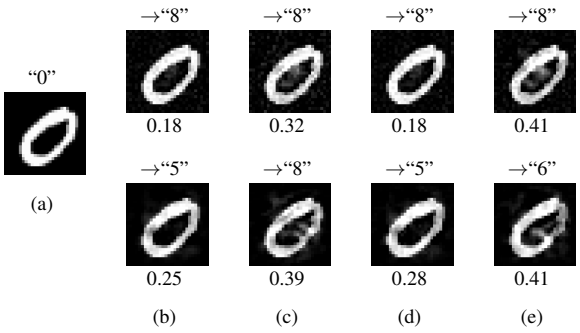

Figure 3: An image ($\mathbf{x}_k$) labelled "0" from the MNIST test set with DeepFool-based adversarial examples generated to fool different models including: (b) the references, (c)-(e): fine-tuned models with adversarial training, Parseval training and our DeepDefense method. Arrows above the pictures indicate which class the examples are "misclassified" to and the numbers below indicate values of $\|\Delta_{\mathbf{x}_k}\|_2/\|\mathbf{x}_k\|_2$. Upper images are generated for MLP models and lower images are generated for LeNet models.

models (*i.e.*, momentum: 0.9, and weight decay: 0.0005).

Robustness and accuracy. The accuracy performance of different methods can be found in the fourth column of Table 1 and the robustness performance is compared in the last five columns. We see DeepDefense consistently and significantly outperforms competitive methods in the sense of both robustness and accuracy, even though our implementation of adversarial training achieves slightly better results than those reported in [26]. Using our method, we obtain an MLP model with **over 2**$\times$ better robustness to DeepFool and **nearly 3**$\times$ better robustness to FGS attack considering ρ, while the inference accuracy also increases a lot (from 98.31% to **98.65%** in comparison with the reference model. The second best is adversarial training, which achieves roughly 1.5$\times$ and 1.8$\times$ improvement to DeepFool and FGS attacks, respectively. Parseval training yields models with improved robustness to the FGS attack, but they are still vulnerable to the DeepFool. The superiority of our method holds on LeNet, and the test-set accuracy increases from 99.02% to **99.34%** with the help of our method.

Convergence curves of different methods are provided in Figure 2, in which the "Clean" curve indicates fine-tuning on the clean training set with the original classification objective. Our method optimizes more sophisticated objective than the other methods so it takes longer to finally converge. However, both its robustness and accuracy performance surpasses that of the reference models in only three epochs and keeps growing in the last two. Consistent with results reported in [26], we also observe growing accuracy and decreasing ρ_2 during adversarial training.

In fact, the benefit of our method to test-set accuracy is

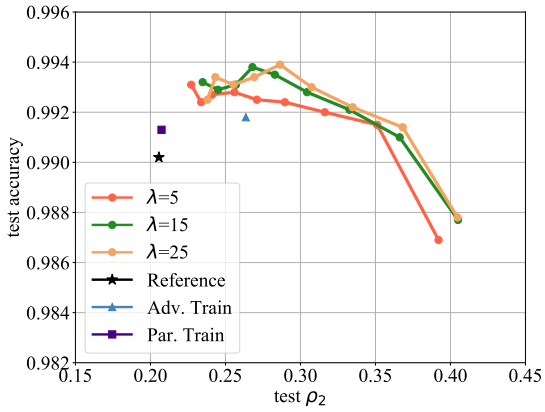

Figure 4: The performance of our DeepDefense with varying hyper-parameters on MNIST. We use LeNet as reference network here. Different points on the same curve correspond to fine-tuning with different values of c (decreasing from left to right). Best viewed in color.

unsurprising. An accurate $\Delta_{\mathbf{x}_k}$ represents the distance from an input sample $\mathbf{x}_k$ to the decision boundary, so maximizing $\|\Delta_{\mathbf{x}_k}\|$ approximately maximize the margin. According to some theoretical works [35, 31], such regularization to the margin should relieve the overfitting problem of complex learning models (including DNNs) and thus lead to better test-set performance.

Visualization Results. Quantitative results in Table 1 demonstrate that an adversary has to generate larger perturbations to successfully attack our DeepDefense models. Intuitively, this implies the required perturbations should be perceptually more obvious when fooling our regularized models. We further provide visualization results in Figure 3. DeepFool is adopted to perform attacks here. Given a clean image from the test set (as illustrated in Figure 3(a)), the generated adversarial examples for successfully fooling different models are shown in Figure 3(b)-3(e). Obviously, our method yields more robust models in comparison with the others, by making the adversarial examples closely resembling real "8" and "6" images. More interestingly, our regularized LeNet model predicts all examples in Figure 3(a)-3(d) correctly as "0". For the lower adversarial example in Figure 3(e), it makes the correct prediction "0" with a probability of 0.30 and the incorrect one (*i.e.*, "6") with a probability of 0.69.

Varying Hyper-parameters. Figure 4 illustrates the impact of the hyper-parameters in our method. We fix $d = 5$ and try to vary c and λ in $\{5, 10, 15, 20, 25, 30, 35, 40, 45\}$ and $\{5, 15, 25\}$, respectively. Note that d is fixed here because it has relatively minor effect on our fine-tuning process on MNIST. In the figure, different solid circles on the

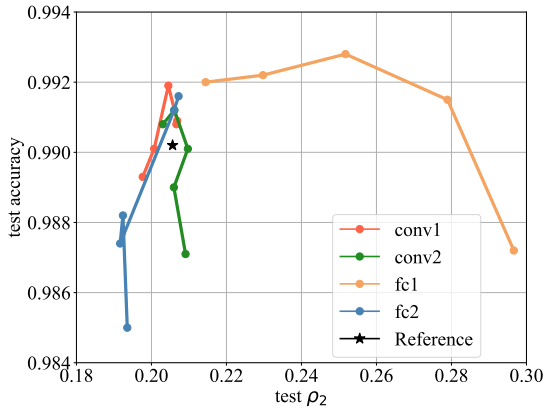

Figure 5: The performance of our method when only one layer is optimized to regularize the LeNet classification objective. Different points on the same curve correspond to different values of c. Legends indicate the gradient of which layer is **not** masked. Best viewed in color.

same curve indicate different values of c. From left to right, they are calculated with decreasing c, which means a larger c encourages to achieve a better accuracy but lower robustness. Conversely, setting a very small c (*e.g.*, $c = 5$) can yield models with high robustness but low accuracies. By adjusting λ, one changes the numerical range of the regularizer. A larger λ makes the regularization contributes more to the whole objective function.

Layer-wise Regularization. We also investigate the importance of different layers to the robustness of LeNet with our DeepDefense method. Specifically, we mask the gradient (by setting its elements to zero) of our adversarial regularizer w.r.t. the learning parameters (*e.g.*, weights and biases) of all layers except one. By fixing $\lambda = 15$, $d = 5$ and varying c in the set $\{5, 15, 25, 35, 45\}$, we obtain 20 different models. Figure 5 demonstrates the ρ_2 values and test-set accuracies of these models. Apparently, when only one layer is exploited to regularize the classification objective, optimizing "fc1" achieves the best performance. This is consistent with previous results that "fc1" is the most "redundant" layer of LeNet [10, 9].

4.3. Image Classification Experiments

For image classification experiments, we testify the effectiveness of our method on several different benchmark networks on the CIFAR-10 and ImageNet datasets.

CIFAR-10 results. The CIFAR-10 dataset is comprised of 60,000 color images (50,000 for training and 10,000 for test) of shape 32×32. We train two CNNs as references: one with the same architecture as in [13], and the other with a network-in-network architecture [21]. Our training procedure is the same as introduced in the MatConvNet exam-

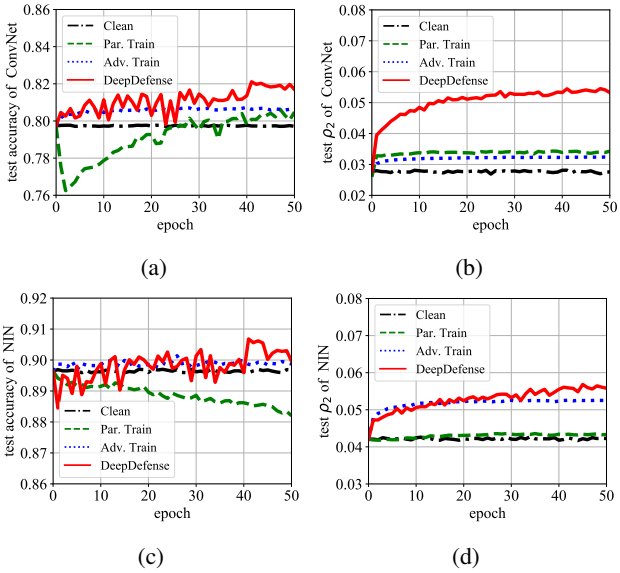

Figure 6: Convergence curves on CIFAR-10: (a) test accuracy of ConvNet, (b) test ρ_2 of ConvNet, (c) test accuracy of NIN, and (d) test ρ_2 of NIN. Best viewed in color.

ples [33]. We still compare our method with adversarial and Parseval training by fine-tuning from the references. Some fine-tuning hyper-parameters are summarized in the third row of Table 2. Likewise, we cut the learning rate by $2\times$ for the last 10 epochs.

Quantitative results can be found in Table 1 and the convergence curves of our method are illustrated in Figure 6, in which the two chosen CNNs are referred to as "ConvNet" and "NIN", respectively. Obviously, our DeepDefense outperforms the other defense methods considerably in all test cases. When compared with the reference models, our regularized models achieve higher test-set accuracies on clean images and **over $2\times$** better robustness to the FGS attack. For the DeepFool attack which is stronger, our method gains $2.1\times$ and $1.3\times$ better robustness on the two networks.

ImageNet results. As a challenging classification dataset, ImageNet contains millions of high-resolution images [30]. To validate the efficacy and scalability of our method, we collect a well-trained AlexNet model [17] from the Caffe Model Zoo [16], fine-tune it on the ILSVRC-2012 training set using our DeepDefense and test it on the validation set. After only 10 epochs of fine-tuning, we achieve **over $1.5\times$** improved robustness to both the DeepFool and FGS attacks along with an increased test-set accuracy, highlighting the effectiveness of our method.

5. Conclusion

In this paper, we investigate the vulnerability of DNNs to adversarial examples and propose a novel method to re-

duce it, by incorporating an adversarial perturbation-based regularization into the classification objective. This endows DNN models with a particular ability of learning from adversarial attacks and resisting them, directly and precisely. We consider the joint optimization problem as learning a recursive-flavoured network and solve it efficiently. Extensive experiments on MNIST, CIFAR-10 and ImageNet have shown the effectiveness of our method. Future works shall include explorations on resisting black-box attacks.

References

[1] Pytorch. https://github.com/pytorch. Accessed: 2017-10-21. 5

[2] Robust vision benchmark. https://robust.vision/benchmark/leaderboard. Accessed: 2017-10-21. 3

[3] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. Deep variational information bottleneck. In *ICLR*, 2017. 2, 5

[4] N. Carlini and D. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *ACM Workshop on Artificial Intelligence and Security*, 2017. 2

[5] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier. Parseval networks: Improving robustness to adversarial examples. In *ICML*, 2017. 1, 2, 5, 6

[6] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard. Robustness of classifiers: from adversarial to random noise. In *NIPS*, pages 1632–1640, 2016. 1

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015. 1, 2, 3, 4, 6

[8] S. Gu and L. Rigazio. Towards deep neural network architectures robust to adversarial examples. In *ICLR Workshop*, 2015. 1, 2, 6

[9] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *NIPS*, pages 1379–1387, 2016. 8

[10] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *NIPS*, pages 1135–1143, 2015. 8

[11] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 4

[12] M. Hein and M. Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NIPS*, 2017. 1, 2

[13] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 8

[14] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017. 2

[15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 4

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *MM*. ACM, 2014. 8

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 4, 8

[18] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016. 2

[19] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In *ICLR*, 2017. 1, 2, 4, 6

[20] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. *Shape, contour and grouping in computer vision*, pages 823–823, 1999. 6

[21] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014. 8

[22] J. Lu, T. Issaranon, and D. Forsyth. Safetynet: Detecting and rejecting adversarial examples robustly. In *ICCV*, 2017. 2

[23] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff. On detecting adversarial perturbations. In *ICLR*, 2017. 2

[24] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *arXiv preprint arXiv:1704.03976*, 2017. 6

[25] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *CVPR*, 2017. 1, 2

[26] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. DeepFool: a simple and accurate method to fool deep neural networks. In *CVPR*, 2016. 1, 2, 3, 4, 5, 6, 7

[27] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015. 4

[28] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical black-box attacks against machine learning. In *Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017. 1, 2

[29] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Symposium on Security and Privacy*, pages 582–597. IEEE, 2016. 2

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 5, 8

[31] J. Sokolic, R. Giryes, G. Sapiro, and M. R. Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 2017. 7

[32] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *ICLR*, 2014. 1, 2, 3, 4, 6

[33] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *MM*. ACM, 2015. 8

[34] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille. Adversarial examples for semantic segmentation and object detection. In *ICCV*, 2017. 2, 3

[35] H. Xu and S. Mannor. Robustness and generalization. *Machine learning*, 86(3):391–423, 2012. 7

[36] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 4

DeepDefense: Training Deep Neural Networks with Improved Robustness Supplementary Material

Ziang Yan[1*] Yiwen Guo[2*] Changshui Zhang[1]
[1]Department of Automation, Tsinghua University
State Key Laboratory of Intelligent Technology and Systems
Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing, China
[2] Intel Labs, China
yza15@mails.tsinghua.edu.cn yiwen.guo@intel.com zcs@mail.tsinghua.edu.cn

1. Unnormalized Norm

In the main body of our paper, we utilize the normalized l_p norm of required adversarial perturbations to evaluate the robustness of different models, as suggested in the DeepFool paper [4]. We notice that in some papers, an unnormalized norm is used instead. For instance,

$$\rho_2' := \frac{1}{|\mathcal{D}|} \sum_{k \in \mathcal{D}} \|\Delta_{\mathbf{x}_k}\|_2 \tag{1}$$

can be calculated to evaluate the robustness of different models to the DeepFool attack. Here we further compare the effectiveness of different defense methods on the base of such metric in Table 1. It can be seen that the significant superiority of our method holds on various baseline networks and different test datasets.

2. ResNet results

We also verify the effectiveness of our DeepDefense on ResNet [2], which is one of the most advanced network architectures at present. We download a pre-trained ResNet-18 model online and fine-tune it on the ILSVRC-2012 training set as with fine-tuning AlexNet. We cut the learning rate by half after 0.3 epochs of fine-tuning and test the result after only 0.5 epochs. We follow the same data augmentation protocol as training the reference model: random crop and horizontal flip, and use center crop for test. The Inference accuracy and robustness performance on the validation set is reported in Table 2. Our method achieve nearly $2\times$ better robustness to both the DeepFool and FGS attacks, and the inference accuracy also increases. Note that the reference shows slightly lower accuracy on our test set than reported, which is probably on account of the different decoding operation of image files. To make it clear, we will report the performance of our method on a ResNet-18 model trained by ourselves (in a future version) if possible.

Dataset	Network	Method	ρ_2'
MNIST	MLP	Reference	206.96
		Par. Train [1]	205.89
		Adv. Train [4]	300.67
		DeepDefense	**418.29**
	LeNet	Reference	378.74
		Par. Train [1]	383.29
		Adv. Train [4]	486.62
		DeepDefense	**524.21**
CIFAR-10	ConvNet	Reference	47.13
		Par. Train [1]	62.16
		Adv. Train [4]	55.48
		DeepDefense	**92.39**
	NIN	Reference	76.95
		Par. Train [1]	79.30
		Adv. Train [4]	95.80
		DeepDefense	**102.61**
ImageNet	AlexNet	Reference	76.08
		DeepDefense	**115.97**

Table 1: Robustness of different defense methods (in the sense of ρ_2' value) under the DeepFool attack.

We also found it is difficult to achieve a good trade-off between robustness and accuracy for Parseval and adversarial training on ImageNet. In our experiments on AlexNet, we were unable to find a suitable β such that the fine-tuned model shows reasonably high accuracy ($> 56\%$) and significantly improved robustness simultaneously for Parseval training. This phenomenon can possibly be caused by insufficient hyper-parameter search, but please note that the original paper [1] does not report ImageNet results as well. For adversarial training, we observed a drastically decrease of inference accuracy when the fine-tuning process starts, and

Dataset	Network	Method	Acc.	ρ_2	ρ_∞	Acc.@$0.2\epsilon_{\text{ref}}$	Acc.@$0.5\epsilon_{\text{ref}}$	Acc.@$1.0\epsilon_{\text{ref}}$
ImageNet	ResNet-18	Reference	66.63%	1.90×10^{-3}	7.56×10^{-4}	60.02%	50.45%	37.66%
		DeepDefense	**66.81%**	$\mathbf{3.68\times10^{-3}}$	$\mathbf{1.46\times10^{-3}}$	**63.25%**	**58.08%**	**50.00%**

Table 2: The performance of our method under adversarial attacks on ImageNet validation set. Column 4: accuracies on clean test images. Column 5: ρ_2 values under the DeepFool attack. Column 6: ρ_∞ values under the Fast Gradient Sign (FGS) attack. Column 7-9: accuracies on the FGS perturbed images, and ϵ_{ref} is chosen as the smallest ϵ such that 50% perturbed images are misclassified by our regularized model.

after 10 epochs the accuracy is still low ($< 50\%$). However, Kurakin *et al*. [3] have produced an Inception v3 model [**?**] using 50 machines after $150k$ iterations (*i.e*. roughly 187 epochs) of training and obtain only slightly degraded accuracy, so we guess more training epochs and sophisticated mixture of clean and adversarial examples are required.

3. Network Architectures

MNIST		CIFAR-10	
MLP	LeNet	ConvNet	NIN
Input (28×28)		Input (32×32)	
FC-500	Conv-5-20	Conv-5-32	Conv-5-192
ReLU	MaxPool-2	MaxPool-2	ReLU
FC-150	Conv-5-50	ReLU	Conv-1-160
ReLU	MaxPool-2	Conv-5-32	ReLU
FC-10	FC-500	ReLU	Conv-1-96
	ReLU	AvgPool-2	ReLU
	FC-10	Conv-5-64	MaxPool-2
		ReLU	Conv-5-192
		AvgPool-2	ReLU
		Conv-4-64	Conv-1-192
		ReLU	ReLU
		Conv-1-10	Conv-1-192
			ReLU
			AvgPool-2
			Conv-3-192
			ReLU
			Conv-1-192
			ReLU
			Conv-1-10
			AvgPool-8

Table 3: Network architectures in MNIST and CIFAR-10 experiments. We use Conv-[kernel width]-[output channel number], FC-[output channel number], MaxPool-[kernel width], AvgPool-[kernel width] to denote parameters of convolutional layers, fully-connected layers, max pooling layers and average pooling layers, respectively.

Table 3 shows the architecture of networks used in our MNIST and CIFAR-10 experiments. For AlexNet experiments, we directly use the reference model from the Caffe model zoo.

References

[1] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier. Parseval networks: Improving robustness to adversarial examples. In *ICML*, 2017. 1

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[3] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In *ICLR*, 2017. 2

[4] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. DeepFool: a simple and accurate method to fool deep neural networks. In *CVPR*, 2016. 1