Generating Adversarial Examples Using Parameter-Free Penalty Method

Jiyuan Sun[1,*], Haibo Yu[2], and Jianjun Zhao[3]
[1]Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan
[2]Department of Information Science, Faculty of Science and Engineering, Kyushu Sangyo University, Fukuoka, Japan
[3]Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan
jiyuan.sun6@gmail.com, yu-haibo@is.kyusan-u.ac.jp, zhao@ait.kyushu-u.ac.jp
*corresponding author

Abstract—Deep neural networks are extremely vulnerable to adversarial examples. An adversarial example is a perturbed input intended to cause deep neural networks to behave incorrectly. Despite much research about the techniques of generating adversarial perturbations, most optimization-based attacks that minimize the perturbations solve a constrained nonlinear programming problem by adding the misclassifying constraint to the objective function using a penalty parameter or a Lagrange multiplier, which is usually found by using binary search. However, this is inefficient due to the many iterations in binary search.

The penalty method is one of the most commonly used iterative methods for solving constrained optimization problems. It solves a constrained problem by converting it to an unconstrained one. In this work, we propose a novel white-box adversarial attack based on a parameter-free penalty method, which could generate adversarial examples with small perturbations at a high success rate. Extensive experiments on three test benches have been conducted, and results show that our method can obtain a good balance between the attack success rate and the perturbation size.

Keywords–adversarial attack; adversarial example; neural network; parameter-free penalty method; perturbation; reliability

1. Introduction

Deep neural networks have been widely used and have shown outstanding performance in various machine learning tasks, such as computer vision, speech recognition, and natural language processing. However, they are also shown to be vulnerable to attacks, posing a threat to the reliability and security of deep learning systems, especially in safety-critical applications such as self-driving. One kind of attack is the adversarial example, which was first discovered in [1] and has received much attention in recent years. To generate adversarial examples, small perturbations are crafted deliberately and added to the inputs, causing misclassifications of the models. These perturbations are generally so small that they are imperceptible to humans. However, they could lead to totally different prediction results. For example, in Figure 1, one image from the validation set of ImageNet and its corresponding adversarial example generated by our method are given. The prediction changes from class 370 for the original image to 371 for its adversarial image.

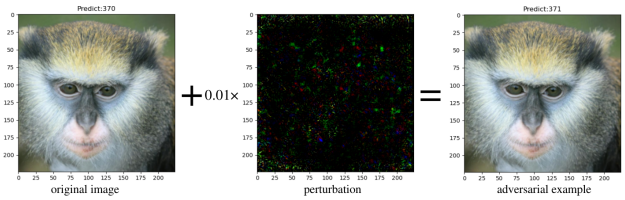

Figure 1: An ℓ_2 adversarial example generated by our method on ImageNet

Similar to bugs and faults in traditional programming, adversarial examples cause totally incorrect predictions of deep neural networks. However, different interpretations are given for the existence of adversarial examples. Goodfellow et al. [2] explain the existence of adversarial examples by the linearity of neural networks. Due to the linearity, the output of deep neural networks grows linearly with respect to the dimensions of the input, and small perturbations in these dimensions together could lead to a significant difference in the output. Szegedy et al. [1] interpret that adversarial examples represent low-probability 'pockets' in the high-dimensional manifold that are hard to find by randomly sampling in the training phase, making them blind spots for neural networks. Ilyas et al.'s interpretation in [3] is that adversarial examples exist because models learn nonrobust features from the training data. All these interpretations provide intuitions for generating adversarial examples and defending models against attacks.

Currently, various methods for generating adversarial examples have been proposed. Optimization-based attacks try to generate adversarial examples by solving a constrained optimization problem. To solve this optimization problem, methods such as L-BFGS [1], CW [4], and EAD [5] need to find a Lagrange multiplier or a penalty parameter to convert the constrained problem to an unconstrained one. To that end, binary search is commonly used in these methods. However, this is inefficient. For example, CW and EAD both use as many as nine iterations of binary search. In this study, we propose an attack that could generate adversarial examples based on the parameter-free penalty method to improve efficiency. The contributions of our work are as follows:

- We propose a novel white-box adversarial attack, PFPMA (Parameter-Free Penalty Method Attack), which could generate ℓ_2 and ℓ_1 adversarial examples. Detailed theoretical derivations and algorithms are given.

- Extensive experiments on MNIST, CIFAR10, and ImageNet are conducted to demonstrate the efficiency of the proposed method. We analyze the properties of the generated adversarial examples and compare the performance with existing methods. The code for the proposed method and the experiments are available at https://github.com/sjysjy1/PFPMA.

The remainder of this paper is organized as follows: Section 2 introduces adversarial examples and the parameter-free penalty method. The derivations and algorithms of our attack are described in detail in Section 3. Experiments for evaluating the proposed attack and corresponding results are presented in Section 4.

2. Preliminary and Related Work

2.1 Adversarial Example

In this section, we introduce the adversarial example and its generating methods. Let the model be $f(\mathbf{x}) : \mathcal{R}^m \rightarrow \mathcal{R}^C$, where m is the dimension of the input and C is the total number of classes, then its prediction for an input $\mathbf{x}_0$ is $\text{argmax}_i \ f_i(\mathbf{x_0})$. A perturbed input $\mathbf{x}_0 + \mathbf{r}$ is called an adversarial example of $\mathbf{x}_0$ if

$$\text{argmax}_i \ f_i(\mathbf{x}_0) \neq \text{argmax}_i \ f_i(\mathbf{x}_0 + \mathbf{r})$$

where $\mathbf{r}$ is a small perturbation and $f_i(\cdot)$ denotes the ith element of $f(\cdot)$. Currently, optimization-based adversarial attacks fall into two categories. The first category contains methods like L-BFGS [1], CW [4], and EAD [5], which minimize the size of adversarial perturbations with respect to the misclassifying constraint and this can be formulated as the following constrained optimization problem:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \|\mathbf{x} - \mathbf{x}_0\|_p \\ \text{subject to} \quad & \underset{i=1,2,\ldots,C}{\arg\max}\{f_i(\mathbf{x})\} \neq l \\ & \mathbf{x} \in [0,1]^m \end{aligned} \tag{1}$$

where l is the correct label for $\mathbf{x}_0$, $||\cdot||_p$ is the ℓ_p norm for vectors and the solution $\mathbf{x}$ is called an ℓ_p adversarial example. This is also the problem the proposed attack in this work tries to solve. This category of attacks generates adversarial examples with minor perturbations but takes a relatively longer time.

The second category is to maximize the loss within a given perturbation budget, which can be formulated as

$$\begin{aligned} \underset{\mathbf{x}}{\text{maximize}} \quad & loss(f(\mathbf{x}), l) \\ \text{subject to} \quad & \|\mathbf{x} - \mathbf{x}_0\|_p \leq \epsilon \\ & \mathbf{x} \in [0,1]^m \end{aligned} \tag{2}$$

where $loss(f(\mathbf{x}, l))$ is a function to measure the distance between outputs of neural networks and expected labels. FGSM [2], PGD [6] attack and all their variants fall into this category, and these attacks generally use fewer iterations to generate adversarial examples, meaning that they are fast. Thus, they are often used in adversarial training. Since adversarial examples generated by this category of attacks have high loss values, they tend to have a better transferability property.

2.2 Related Work

There is extensive research regarding the generation of adversarial examples based on solving problem (1) or (2). In this section, we briefly introduce attacks that are close to the method proposed in this work.

L-BFGS attack [1] solves the targeted version of problem (1) by first adding the constraint to the objective via a multiplier, yielding

$$\underset{\mathbf{x}}{\text{minimize}} \ c|\mathbf{r}| + loss(f(\mathbf{x}), l_{tar}) \text{ subject to } \mathbf{x} \in [0,1]^m \tag{3}$$

where c is found by binary search, solving the problem with the L-BFGS method.

Similarly, CW attack [4], which is currently one of the most commonly used attacking methods, first reformulates the misclassifying constraint into "$L_{CW}(f(\mathbf{x}), l) \leq 0$" form and then adds it to the objective function, yielding

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & c \cdot ||\mathbf{x} - \mathbf{x_0}||_p + L_{CW}(f(\mathbf{x}, l)) \\ \text{subject to} \quad & \mathbf{x} \in [0,1]^m \end{aligned} \tag{4}$$

where, like L-BFGS, c is also found by binary search. CW attack uses some involving techniques to perform $\ell_0, \ell_2, \ell_\infty$ attacks.

EAD [5] tries to find adversarial examples with both small ℓ_1 and ℓ_2 perturbation by solving the following optimization problem:

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimize}} \ loss(f(\mathbf{x}), l) + \lambda_1||\mathbf{x} - \mathbf{x_0}||_1 + \lambda_2||\mathbf{x} - \mathbf{x_0}||_2^2 \\ &\text{subject to } \mathbf{x} \in [0,1]^m \end{aligned} \tag{5}$$

where λ_1, λ_2 are regularization parameters for ℓ_1 and ℓ_2 penalty terms, respectively, the binary search is used for them. EAD degenerates to an ℓ_1 attack if $\lambda_2 = 0$.

FGSM [2] is an ℓ_∞ attack using the following single step

$$\begin{aligned} \eta &= \epsilon \text{sign}(\nabla_{\mathbf{x}} loss(\theta, \mathbf{x}, l)) \\ \mathbf{x} &= \mathbf{x}_0 + \eta \end{aligned} \tag{6}$$

Although FGSM is fast, its attack success rate is not satisfactory. Thus, several iterative variants of FGSM have been proposed in [6] [7] [8], and these attacks differ in the methods for solving problem (2).

DDN [9], which solves an adaptive form of problem (1), iteratively moves in the gradient direction and checks whether the current point is an adversarial example to determine whether to decrease or increase the perturbation. Finally, the attack is expected to obtain adversarial examples with minor noise. FMN [10] extends the ℓ_2 DDN attack to $\ell_0, \ell_1, \ell_\infty$ attacks.

DeepFool [11] is a boundary-based ℓ_2 attack that generates adversarial examples by first linearizing the model function at each iterative point and then updating the point with its projection to the approximated decision boundary, which has a closed-form expression. It focuses on finding adversarial examples with perturbations that are as small as possible. FAB [12] is a variant of DeepFool attack, which tries to update the iterative point using a convex combination of the projection

and the original input, making the adversarial example closer to the original image than DeepFool.

2.3 Penalty Method and Parameter-Free Penalty Method

2.3.1 Penalty Method

As the optimization problem for the adversarial attack has only one inequality constraint, ignoring the box constraint $\mathbf{x} \in [0,1]^m$, we only introduce the penalty method for problems with inequality constraints. For a nonlinear programming problem of the following form

$$\begin{aligned} &\text{minimize} \quad f(\mathbf{x}) \\ &\text{subject to} \quad g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, ..., n \end{aligned} \tag{7}$$

penalty method eliminates some or all of the constraints and adds corresponding penalty terms to the objective function, obtaining the penalty problem below:

$$\text{minimize} \quad f(\mathbf{x}) + \sum_{i=1}^{n} \mu_i P_i(\mathbf{x}) \tag{8}$$

where $\mu_i > 0$, $P_i(\mathbf{x}), i = 1, 2, ..., n$ are the penalty parameters and penalty functions respectively. $P_i(\mathbf{x}) > 0$ if $g_i(\mathbf{x}) > 0$, otherwise $P_i(\mathbf{x}) = 0$. In other words, it will penalize $\mathbf{x}$ that is not feasible. The penalty method solves this problem using any technique for unconstrained problems. If μ_i is positive infinity, this penalty problem has the same solution as the original problem. However, in practice, several iterations are required, and μ_i increases in each iteration since a large penalty parameter would cause ill-conditioning, due to which the optimization process may become unstable. Thus, μ_i should be chosen carefully.

2.3.2 Parameter-Free Penalty Method

To improve efficiency and avoid the ill-conditioning problem, the parameter-free penalty method tries to find a better solution than the one obtained in the previous iteration. Its steps for solving problem (7) can be summarized as follows:

step 1: Find a lower bound B_1 for the objective function, which can be formally written as $B_1 \leq \textbf{inf}\{f(\mathbf{x}) : g_i(\mathbf{x}) \leq 0, i = 1, 2, 3, .., m\}$

step 2: Formulate a penalty problem of the following form for (7):

$$\min_{\mathbf{x}} \; P_1(f(\mathbf{x}) - L_k) + \sum_{i=1}^{n} \mu_i P_2(g_i(\mathbf{x})) \tag{9}$$

where $P_1(\cdot), P_2(\cdot)$ are any penalty functions and solve it to find its optimal solution $\mathbf{x}_k$.

step 3: Update the bound as follows and then proceed to step 2 until the maximum iteration number is reached:

$$L_k = f(\mathbf{x_k})$$

If the final $\mathbf{x}_k$ is an optimal solution of the penalty problem and also feasible for the original problem (7), then $\mathbf{x}_k$ is an optimal solution for the original problem [13].

3. Methodology

Generally, the parameter c and λ_1, λ_2 in (3), (4), and (5) for L-BFGS, CW, and EAD attacks vary for different inputs. To find these parameters, methods like L-BFGS, CW, and EAD use binary search, which is inefficient. Boundary-based attacks like DeepFool and FAB don't use the penalty parameters. However, they need to construct linear approximations for each class, making their complexity property notorious for large data sets such as ImageNet. The parameter-free penalty method doesn't need to find a penalty parameter by binary search, and its complexity doesn't increase with the dimensions of data sets or the number of classes. With these advantages, we expect the parameter-free penalty method to perform well in adversarial attacks. Each iteration penalizes the solutions that are worse than previous iterations. Theoretically, if the optimal solution of the final penalty subproblem is also feasible for the original optimization problem, it's an optimal solution for the attacking problem. However, in practice, the optimal solution to the subproblem is probably not easy to find. Thus, we regard the penalty term as a regularization with a fixed parameter and use a bound that is smaller than $f(\mathbf{x})$ instead of exactly $f(\mathbf{x})$. The attack is introduced below in detail.

Following [4], we first replace the misclassifying constraint $\arg\max_{i=1,2,...,C}\{f_i(\mathbf{x})\} \neq l$, which is highly indifferentiable, with the following inequality constraint:

$$L(\mathbf{x}, l) \overset{\text{def}}{=} -\max_{i \neq l} f_i(\mathbf{x}) + f_l(\mathbf{x}) \leq 0 \tag{10}$$

and then we have the following problem

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \|\mathbf{x} - \mathbf{x}_0\|_p \\ \text{subject to} \quad & L(\mathbf{x}, l) \leq 0 \\ & \mathbf{x} \in [0,1]^m \end{aligned} \tag{11}$$

for which we formulate its penalty problem as

$$\begin{aligned} &\min_{\mathbf{x}} \; (P_1(\mathbf{x}) + \lambda P_2(\mathbf{x})) \\ &\text{subject to } \mathbf{x} \in [0,1]^m \end{aligned} \tag{12}$$

Since the objective value is non-negative, we can use $B_1 = 0$.

3.1 ℓ_2 attack

We introduce the ℓ_2 attack first as it's less challenging due to the differentiability of the objective function. For the simplicity of calculating gradient, we alternatively minimize $||\mathbf{x}||_2^2$ instead of $||\mathbf{x}||_2$ and use the same penalty function for both the objective and the constraint. For the case of the max penalty function, the penalty problem is

$$\begin{aligned} &\min_{\mathbf{x}} \; \big(\max\{0, ||\mathbf{x}||_2^2 - B\} + \lambda \max\{0, L(\mathbf{x}, l)\}\big) \\ &\text{subject to } \mathbf{x} \in [0,1]^m \end{aligned} \tag{13}$$

and the gradients for the two terms in the objective are

$$\nabla_1 = \begin{cases} 2\mathbf{x}, & \text{if } ||\mathbf{x}||_2^2 - B > 0 \\ 0, & \text{else} \end{cases} \tag{14}$$

$$\nabla_2 = \begin{cases} \lambda \nabla_{\mathbf{x}} L(\mathbf{x}, l), & \text{if } L(\mathbf{x}, l) > 0 \\ 0, & \text{else} \end{cases} \tag{15}$$

In the case of the max-square penalty function, the penalty problem is

$$\begin{aligned} &\min_{\mathbf{x}} \ \left(\max\{0, ||\mathbf{x}||_2^2 - B\}^2 + \lambda \max\{0, L(\mathbf{x}, l)\}^2\right) \\ &\text{subject to } \mathbf{x} \in [0,1]^m \end{aligned} \tag{16}$$

whose gradients for the two terms in the objective are

$$\nabla_1 = \begin{cases} 4(||\mathbf{x}||_2^2 - B)\mathbf{x}, & \text{if } ||\mathbf{x}||_2^2 - B > 0 \\ 0, & \text{else} \end{cases} \tag{17}$$

$$\nabla_2 = \begin{cases} 2\lambda L(\mathbf{x}, l) \nabla_{\mathbf{x}} L(\mathbf{x}, l), & \text{if } L(\mathbf{x}, l) > 0 \\ 0, & \text{else} \end{cases} \tag{18}$$

The gradient descent method could be used directly to solve these two problems as the two terms of the objective function are both differentiable. To improve the convergence property, in each iteration, we use RMSProp algorithm [14] to adjust the step size automatically as follows:

$$R = \rho R + (1 - \rho) \nabla_{\mathbf{x}}$$
$$\alpha = 1/(\sqrt{R} + \sigma)$$

where α is the step size of the current iteration, rho is the decaying parameter, initial $R = 0$, and $sigma$ is a constant for numerical stability, such as $10 - 8$. Thus, dimensions with a larger accumulated partial derivative would use a smaller step size to move forward. The complete algorithm for this attack is given in Algorithm 1. Since the feasibility of $\mathbf{x}$ is not maintained in the searching process, we use $\mathbf{y}$ to store the best adversarial example ever found, as shown in lines 9 to 10.

3.2 ℓ_1 attack

In this case, for the simplicity of calculating proximal mapping, we only use the max penalty function for the norm term and the max, max-square penalty function for the misclassifying constraint. Thus, we have the following two penalty problems

$$\begin{aligned} &\min_{\mathbf{x}} \ \left(\max\{0, ||\mathbf{x} - \mathbf{x}_0||_1 - B\} + \lambda \max\{0, L(\mathbf{x}, l)\}\right) \\ &\text{subject to } \mathbf{x} \in [0,1]^m \end{aligned} \tag{19}$$

and

$$\begin{aligned} &\min_{\mathbf{x}} \ \left(\max\{0, ||\mathbf{x} - \mathbf{x}_0||_1 - B\} + \lambda \max\{0, L(\mathbf{x}, l)\}^2\right) \\ &\text{subject to } \mathbf{x} \in [0,1]^m \end{aligned} \tag{20}$$

whose gradients for the penalty terms in their objective functions are the same as (15) and (18) respectively.

For both these two cases, if $||\mathbf{x} - \mathbf{x}_0||_1 - B \geq 0$, we use the update of gradient descent. If $||\mathbf{x} - \mathbf{x}_0||_1 - B < 0$, the proximal gradient method is used, whose update at each inner iteration is:

$$\begin{aligned} \mathbf{x} &= \mathbf{x} - \alpha * \nabla_2 \\ \mathbf{x} &= \text{prox}_{\alpha||x||_1}(\mathbf{x}) \end{aligned} \tag{21}$$

Algorithm 1 untargeted ℓ_2 attack of PFPMA

Input: original image $\mathbf{x}_0$, model $f(\mathbf{x})$, correct label l,
Parameter: initial step size α_0,RMSProp decaying parameter ρ, regularization parameter λ, bound decaying parameter γ
Initialization: $\mathbf{y} \leftarrow \mathbf{x}_0$,$R \leftarrow 0$, $B \leftarrow 0$
Output: image $\mathbf{y}$

1: **while** $i < N_1$ **do** ▷ outer loop
2: $k \leftarrow 0$
3: **while** $k < N_2$ **do** ▷ inner loop
4: $\nabla_{\mathbf{x}} \leftarrow \nabla_1 + \nabla_2$ ▷ by (14)(15)(17)(18)
5: $R \leftarrow \rho R + (1 - \rho) * \nabla_{\mathbf{x}}^2$ ▷ gradient mean square
6: $\alpha \leftarrow 1/(\sqrt{R} + \sigma)$ ▷ step size
7: $\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla L(\mathbf{x}, l)$
8: $\mathbf{x} \leftarrow \text{clamp}\{\mathbf{x}, \text{min} = 0, \text{max} = 1\}$
9: **if** $f(\mathbf{x}) \neq l$ **and** $||\mathbf{x} - \mathbf{x_0}||_2 < ||\mathbf{y} - \mathbf{x_0}||_2$ **then**
10: $\mathbf{y} \leftarrow \mathbf{x}$ ▷ store the best $\mathbf{x}$ ever found
11: **end if**
12: **end while**
13: $B \leftarrow \gamma ||\mathbf{x}||_2^2$
14: **end while**
15: **return** $\mathbf{y}$

where $\text{prox}_f(\cdot)$ is the proximal mapping, and the update could be written as

$$\begin{aligned} \mathbf{x} &= \mathbf{x} - \alpha * \nabla_2 \\ x_i &= \begin{cases} x_i - \alpha, & \text{if } x_i - (x_0)_i \geq \alpha \\ (x_0)_i, & \text{if } |x_i - (x_0)_i| < \alpha \\ x_i + \alpha, & \text{if } x_i - (x_0)_i \leq -\alpha \end{cases} \end{aligned} \tag{22}$$

$i = 1, 2, ..., m$, where $x_i, (x_0)_i$ are the ith entry of $\mathbf{x}, \mathbf{x}_0$ respectively and its complete derivation can be found in [15], where this update is referred to as ISTA (Iterative Shrinkage-Thresholding Algorithm). An intuitive interpretation is that ISTA first moves in the negative gradient direction of the penalty term for the constraint and then tries to decrease each entry of $\mathbf{x}$ by an amount of the step size to reduce the ℓ_1 norm. The complete algorithm for ℓ_1 attack is given in Algorithm 2. Similar to the ℓ_2 case, the RMSProp algorithm is also used, but only the gradient of the penalty function for the constraint is used for calculating R, whose update is:

$$R = \rho R + (1 - \rho) \nabla_1$$

$$\alpha = 1/(\sqrt{R} + \sigma)$$

Finally, to satisfy the valid image constraint $\mathbf{x} \in [0,1]^m$, each dimension of $\mathbf{x}$ is clamped between 0 and 1.0 for both ℓ_2, ℓ_1 threat model. All these analyses can be extended to targeted attacks with label l_t by simply replacing (10) with the following constraint

$$L(\mathbf{x}, l_t) = \max_{i \neq l_t} f_i(\mathbf{x}) - f_{l_t}(\mathbf{x}) \leq 0$$

Algorithm 2 untargeted ℓ_1 attack of PFPMA

Input: original image $\mathbf{x}_0$, model $f(\mathbf{x})$, correct label l,
Parameter: initial step size α_0,RMSProp decaying parameter ρ, bound B, regularization parameter λ, bound decaying parameter γ
Initialization: $\mathbf{y} \leftarrow \mathbf{x}_0$, $R \leftarrow 0$, $B \leftarrow 0$
Output: image $\mathbf{y}$

while $i < N_1$ **do** ▷ outer loop
  $k \leftarrow 0$
  while $k < N_2$ **do** ▷ inner loop
    $R \leftarrow \rho R + (1-\rho) * \nabla_1$ ▷ gradient mean square
    $\alpha \leftarrow 1/(\sqrt{R} + \sigma)$ ▷ step size
    $\nabla_2 \leftarrow$ (15) or (18)
    $\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla_2$
    if $||\mathbf{x}||_1 - B > 0$ **then**
      $\mathbf{x} \leftarrow \text{ISTA}(\mathbf{x})$ ▷ by (22)
    end if
    $\mathbf{x} \leftarrow \text{clamp}\{\mathbf{x}, \min = 0, \max = 1\}$
    if $f(\mathbf{x}) \neq l$ **and** $||\mathbf{x} - \mathbf{x_0}||_1 < ||\mathbf{y} - \mathbf{x_0}||_1$ **then**
      $\mathbf{y} \leftarrow \mathbf{x}$ ▷ store the best $\mathbf{x}$ ever found
    end if
  end while
  $B \leftarrow \gamma ||\mathbf{x}||_1$
end while
return $\mathbf{y}$

4. EXPERIMENTS

4.1 Setup

Data sets: To test the performance of our attack and the baselines, three most well-known data sets are used: MNIST, CIFAR10 [16] and ImageNet(ILSVRC2012) [17]. MNIST contains 10-class 28×28 images of hand-written digits, and we use all the 10,000 images in its test set. CIFAR10 has ten classes of RGB images of size $3 \times 32 \times 32$, and in our experiments, the first 1,000 images in its test set are used. ImageNet contains 1,000 classes of RGB images, and we resize the first 1,000 images in its validation set to shape $3 \times 224 \times 224$ for our experiments.

Models: For MNIST, we train a LeNet5, whose structure is illustrated in Table 1, and its accuracy is 98.51%. We also use two modified LeNet5 models from [9], both of which have the structure shown in Table 2. The accuracy values of these two models are 99.44% and 99.02% respectively[1]. For the CIFAR10 data set, we use three models from robustbench [18], denoting Standard-C, WongLinf-C, and WangL2-C. Standard-C is a typically trained non-defensive WideResNet-28-10 with an accuracy of 94.78% on the test set of CIFAR10. WongLinf-C is an ℓ_∞-adversarially trained model form [19], having an accuracy of 83.34%. WangL2-C is an ℓ_2-adversarially trained WideResNet-70-16 from [20] and its accuracy is 95.54%. For ImageNet, we also use three models from robustbench [18], denoting Standard-I, WongLinf-I, and SalmanLinf-I. Standard-I is a non-defensive ResNet-50 with an accuracy of 76.30%. WongLinf-I is an ℓ_∞-adversarially trained ResNet-50 form [19] and has an accuracy of 53.70%. SalmanLinf-I is an ℓ_∞-adversarially trained WideResNet-50-2 from [21] and has an accuracy of 68.20%. The accuracy of these three models is obtained by testing the first 1,000 images in the validation set of ImageNet.

Table 1. LeNet5 model for MNIST

Layer	Size
Convolution + ReLU	3×3×32
Max Pooling	2×2
Convolution + ReLU	3×3×64
Max Pooling	2×2
Fully Connected + ReLU	200
Softmax	10

Table 2. Improved LeNet5 model for MNIST

Layer	Size
Convolution + ReLU	3×3×32
Convolution + ReLU	3×3×32
Max Pooling	2×2
Convolution + ReLU	3×3×64
Convolution + ReLU	3×3×64
Max Pooling	2×2
Fully Connected + ReLU	200
Fully Connected + ReLU	200
Softmax	10

Evaluation metric: The implementation of our attack and all the experiments are based on the PyTorch framework. CW, DeepFool, FAB attack from Torchattack package [22][2] and DDN of Adversarial-library package [23][3] are used as the baselines of the ℓ_2 attack. EAD in Torchattacks, FMN, and FAB in Adversarial-library are used as baselines of ℓ_1 attack. We compare their performance using the metrics of ASR(Attack Success Rate), perturbation size, and runtime on a computer with a GTX3060(12GB) GPU. As the perturbation increases, the ASR will finally reach 100.00%, that is, there is a trade-off between them. Thus, we should consider both ASR and perturbation size when evaluating the performance of attacks instead of comparing one of them singly.

Parameter: For high efficiency, we generate adversarial examples for a batch of input simultaneously, and the batch sizes for MNIST, CIFAR, and ImageNet are 512, 128, and 64, respectively. In all the PFPMA experiments, we use $N_1 = 3$ inner iterations and $N_2 = 150$ outer iterations. The ℓ_2, ℓ_1 attacks on MNIST use a step size 0.01. On CIFAR10 and ImageNet, the step size for defensive models is 0.01, and for non-defensive models, 0.001. The regularization parameter λ

[1]These two models are available at https://github.com/jeromerony/augmented_lagrangian_adversarial_attacks

[2]https://github.com/Harry24k/adversarial-attacks-pytorch

[3]https://github.com/jeromerony/adversarial-library

is chosen from $\{0.1, 1, 10, 100, 1000\}$, and defensive models always use larger λ as more significant perturbations are required to attack them. We use 0.3 as the decaying factor for all attacks' penalty bound γ. The decaying parameter for RMSProp is $\rho \in \{0.99, 0.999\}$.

For a fair comparison, we also tune the parameters of baseline attacks to obtain their best performance. Although in [10], α_0 of FMN is chosen from $\{1, 10, 100\}$, we found that for the models in our experiments, ASR is low if a high α_0 is used. Thus we use $\alpha_0 \in \{0.01, 0.1, 1, 10\}$ and $\gamma_0 \in \{0.05, 0.3\}$ and then select the best result. For DDN, we use 1000 iterations and an initial perturbation norm of 1.0, and the factor for modifying the norm is 0.05, which are all the same as [9]. For CW, we use 10,000 iterations, $\kappa = 0$, and penalty parameter $c \in \{1, 10\}$ and report the best result; the step size is 0.01 for MNIST and 0.001 for CIFAR and ImageNet. For EAD, we use nine iterations of binary search and polynomial decaying learning rate with an initial step size value of 0.01.

Table 3. Results of untargeted ℓ_2 attack on MNIST.

Attack	Model	ASR(%)	Pert.	Time(s)
DeepFool (3000)	LeNet5-M	98.10%	0.844	776.3
	Standard-M	99.70%	1.828	388.3
	DDN-M	99.72%	4.101	242.5
FAB (200)	LeNet5-M	98.92%	0.662	84.5
	Standard-M	100.00%	1.389	400.4
	DDN-M	99.99%	2.685	398.4
CW (10000)	LeNet5-M	97.14%	1.421	430.7
	Standard-M	96.77%	1.800	2112.0
	DDN-M	92.46%	3.149	2327.2
DDN (1000)	LeNet5-M	99.99%	0.631	28.9
	Standard-M	99.99%	1.398	170.7
	DDN-M	99.99%	2.626	169.5
Our-M (3×150))	LeNet5-M	**100.00%**	**0.707**	**20.8**
	Standard-M	**100.00%**	**1.404**	**113.2**
	DDN-M	**100.00%**	**2.689**	**115.3**
Our-MS (3×150))	LeNet5-M	99.56%	0.674	20.6
	Standard-M	95.40%	1.365	112.4
	DDN-M	97.49%	2.749	115.67

Table 4. Results of untargeted ℓ_1 attack on MNIST.

Attack	Model	ASR(%)	Pert.	Time(s)
EAD (1000)	LeNet5-M	100.00%	4.880	113.7
	Standard-M	100.00%	10.081	753.3
	DDN-M	100.00%	16.092	1145.2
FAB (1000)	LeNet5-M	97.59%	3.649	210.4
	Standard-M	99.92%	6.789	1093.0
	DDN-M	99.89%	16.242	1074.2
FMN (1000)	LeNet5-M	99.09%	4.998	32.6
	Standard-M	99.87%	6.990	181.9
	DDN-M	99.88%	17.091	179.4
Our-M (3×150)	LeNet5-M	**100.00%**	**4.465**	**31.2**
	Standard-M	100.00%	7.785	125.7
	DDN-M	97.36%	24.592	125.2
Our-MS (3×150)	LeNet5-M	99.80%	4.350	30.2
	Standard-M	**100.00%**	**7.361**	**123.5**
	DDN-M	99.69%	18.909	124.5

Table 5. Results of untargeted ℓ_2 attack on CIFAR10.

Attack	Model	ASR(%)	Pert.	Time(s)
DeepFool (3000)	Standard-C	92.30%	0.225	498.6
	WangL2-C	98.42%	4.325	115.1
	WongLinf-C	92.64%	1.220	62.9
FAB (200)	Standard-C	100.00%	0.112	5922.8
	WangL2-C	100.00%	1.224	6247.3
	WongLinf-C	100.00%	0.816	640.3
CW (10000)	Standard-C	100.00%	0.124	11005.6
	WangL2-C	93.78%	1.131	37262.0
	WongLinf-C	100.00%	0.829	4318.6
DDN (1000)	Standard-C	99.47%	0.124	2518.3
	WangL2-C	91.36%	1.225	2713.0
	WongLinf-C	94.89%	0.831	277.6
Our-M (3×150))	Standard-C	**100.00%**	**0.158**	**1849.6**
	WangL2-C	95.89%	1.208	1893.3
	WongLinf-C	**100.00%**	**0.861**	**201.4**
Our-MS (3×150))	Standard-C	100.00%	0.198	1835.3
	WangL2-C	96.84%	1.173	1879.9
	WongLinf-C	97.15%	1.118	200.9

Table 6. Results of untargeted ℓ_1 attack on CIFAR10.

Attack	Model	ASR(%)	Pert.	Time(s)
EAD (1000)	Standard-C	100.00%	1.495	9936.4
	WangL2-C	100.00%	22.591	10044.0
	WongLinf-C	100.00%	8.620	986.3
FAB (200)	Standard-C	91.03%	1.893	2766.84
	WangL2-C	85.04%	17.56	3068.3
	WongLinf-C	90.50%	7.364	328.9
FMN (1000)	Standard-C	73.84%	1.428	2469.3
	WangL2-C	76.92%	16.442	2672.9
	WongLinf-C	84.32%	7.537	288.2
Our-M (3×150)	Standard-C	100.00%	1.720	1796.4
	WangL2-C	98.63%	19.611	1871.3
	WongLinf-C	**100.00%**	**8.142**	**196.1**
Our-MS (3×150)	Standard-C	**100.00%**	**1.682**	**1775.0**
	WangL2-C	97.89%	17.950	1871.4
	WongLinf-C	94.54%	7.763	196.2

Table 7. Results of untargeted ℓ_2 attack on ImageNet.

Attack	Model	ASR(%)	Pert.	Time(s)
DeepFool (100)	Standard-I	100.00%	0.214	31209.8
	WongLinf-I	100.00%	1.986	28353.4
	SalmanLinf-I	100.00%	2.690	46637.2
CW (10000)	Standard-I	100.00%	0.213	10175.0
	WongLinf-I	97.58%	1.425	28660.3
	SalmanLinf-I	93.99%	1.728	66307.1
DDN (1000)	Standard-I	67.63%	0.233	2829.7
	WongLinf-I	80.45%	1.606	1983.7
	SalmanLinf-I	79.91%	1.927	4470.0
Our-M $(3 \times 150))$	Standard-I	100.00%	0.243	1823.4
	WongLinf-I	98.51%	1.836	1303.0
	SalmanLinf-I	97.80%	2.649	3052.0
Our-MS $(3 \times 150))$	Standard-I	**100.00%**	**0.224**	**1835.4**
	WongLinf-I	99.07%	3.141	1291.7
	SalmanLinf-I	97.21%	4.490	2982.2

*FAB L2 is not used as it consistently crashed on ImageNet .

Table 8. Results of untargeted ℓ_1 attack on ImageNet.

Attack	Model	ASR(%)	Pert.	Time(s)
EAD (1000)	Standard-I	100.00%	8.837	9168.0
	WongLinf-I	100.00%	31.354	6855.5
	SalmanLinf-I	100.00%	41.506	16585.3
FAB (50)	Standard-I	23.98%	4.385	77185.9
	WongLinf-I	57.17%	30.988	54352.2
	SalmanLinf-I	53.37%	54.006	121073.0
FMN (1000)	Standard-I	49.41%	10.934	2915.2
	WongLinf-I	39.11%	37.902	2050.9
	SalmanLinf-I	40.18%	52.727	4540.0
Ours-M (3×150)	Standard-I	**100.00%**	**11.136**	**1915.2**
	WongLinf-I	**100.00%**	**32.585**	**1323.8**
	SalmanLinf-I	99.27%	49.334	3038.0
Ours-MS (3×150)	Standard-I	99.21%	9.723	1876.4
	WongLinf-I	99.63%	26.615	1324.3
	SalmanLinf-I	97.95%	38.096	3037.8

4.2 Results

We summarize the result of MNIST in Table 3, 4, CIFAR in Table 5, 6, ImageNet in Table 7, 8, where Our-M and Our-MS denote the attacks using max and max-square penalty function respectively. We highlight the attacks that have 100.00% ASR, small perturbations without a significant increase of computation cost, i.e., the attacks that give a good trade-off for these three metrics. It can be observed that although the CW attack achieves a relatively high ASR and a small perturbation size for all three data sets, its runtime is generally an order of magnitude larger than the fastest attacks. DeepFool and FAB perform well on small data sets of MNIST and CIFAR10. Still, they don't have competitive performance in computational efficiency for large data sets because they have to find the linear approximation of the decision boundary for each class, which is relatively inefficient for ImageNet, which has 1,000 categories of images. For example, the runtime of DeepFool is about 20 times larger than that of our method on ImageNet. In comparison, across many cases, our method could achieve a 100.00% success rate with a slightly larger perturbation at high speed, meaning that it has a good balance in these metrics. It also has a suitable complexity property with respect to the number of classes and dimensions.

For visualization, we give some adversarial examples generated for each data set in Figure 2 to Figure 7. For MNIST, the 1st row has the first ten original images correctly predicted by LeNet5-M. Adversarial examples generated by our method using max penalty and max-square penalty are in the 2nd and 3rd row, respectively. Adversarial examples for Standard-M are in the 4th and 5th row and for DDN-M in the 6th and 7th row. As expected, ℓ_2 adversarial examples have small perturbations that are added to a large number of pixels. On the contrary, the ℓ_1 attack tends to generate adversarial perturbations that are sparse but strong on each pixel. We can see that the adversarial perturbations on the DDN-M model are much more noticeable, proving the defense's effectiveness. For CIFAR10, the first ten images correctly predicted by Standard-C are in the 1st row, and their adversarial examples are in the 2nd and 3rd rows. Adversarial examples for WangL2-C are in the 4th and 5th row, and WongLinf-C is in the 6th and 7th row. Compared with MNIST, the perturbations for these RGB images are more imperceptible to humans, meaning that these images are easier to attack. Our interpretation is that larger data sets have more dimensions, and their models tend to have more parameters, allowing a more extensive space for exploiting the attack. Similarly, for ImageNet, the first ten images correctly predicted by Standard-I are in the 1st row, and their adversarial examples are in the 2nd and 3rd rows. Adversarial examples for WongLinf-I are in the 4th and 5th row, and SalmanLinf-I are in the 6th and 7th row.

To study the distribution of perturbations, we plot the curve of the ASR as the perturbation size increases in Figure 8 to Figure 10. A steep slope means that the perturbations are more concentrated. Thus, we can find that the variance of ℓ_1 adversarial perturbation is larger than ℓ_2. The reason is that ℓ_1 adversarial perturbation is sparse and tends to have tremendous values for some pixels. We can also find that large data sets have higher variances. This also makes sense since the norm is the sum of perturbations on all dimensions, and as the dimension increases, the difference between different norms becomes more significant.

5. Conclusion

In this work, we propose a novel white-box adversarial attack based on a parameter-free penalty method and conduct extensive experiments to evaluate its performance. The results show that the proposed method performs competitively with

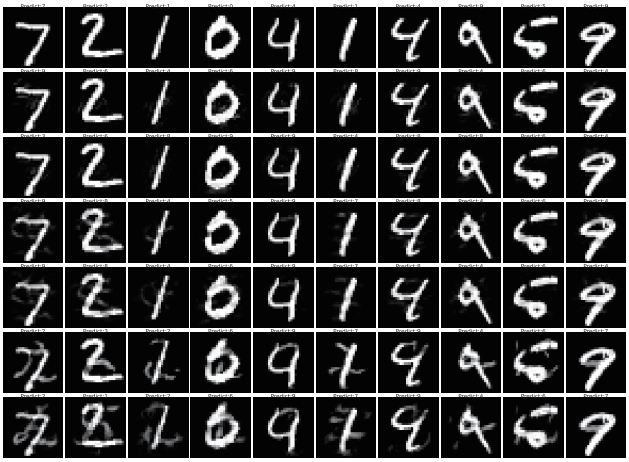

Figure 2: ℓ_2 adversarial examples generated on MNIST

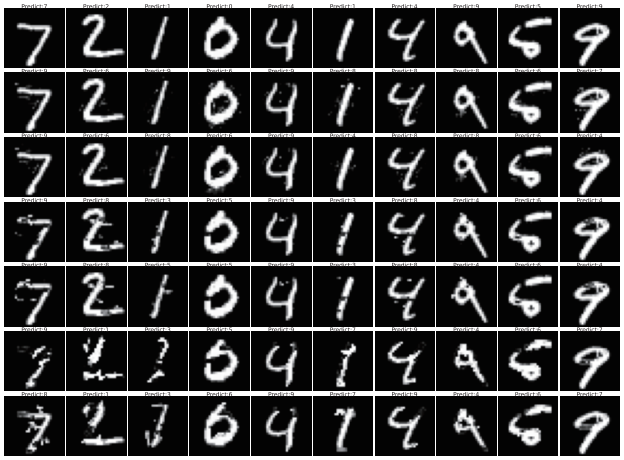

Figure 3: ℓ_1 adversarial examples generated on MNIST

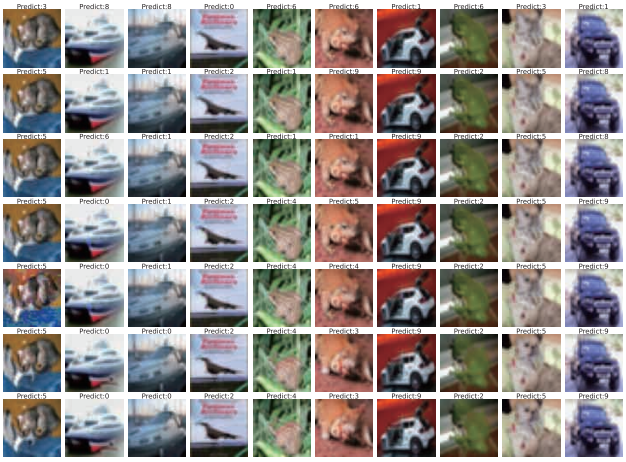

Figure 4: ℓ_2 adversarial examples generated on CIFAR10

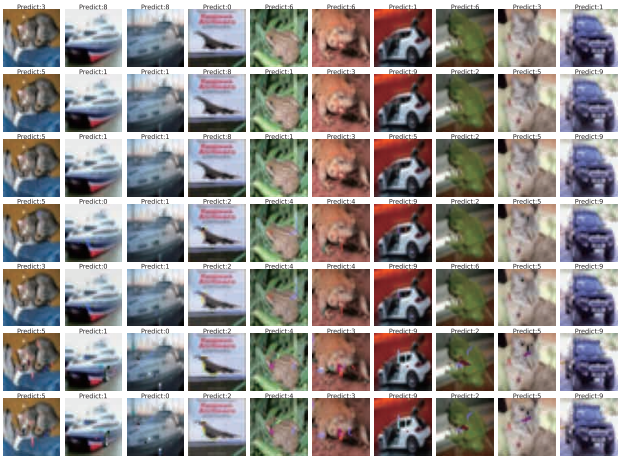

Figure 5: ℓ_1 adversarial examples generated on CIFAR10

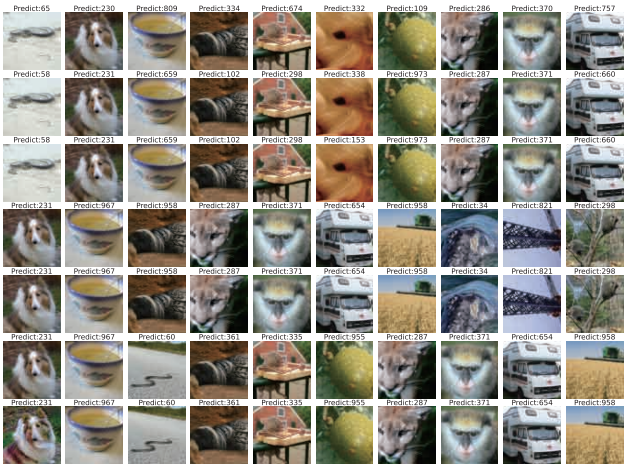

Figure 6: ℓ_2 adversarial examples generated on ImageNet

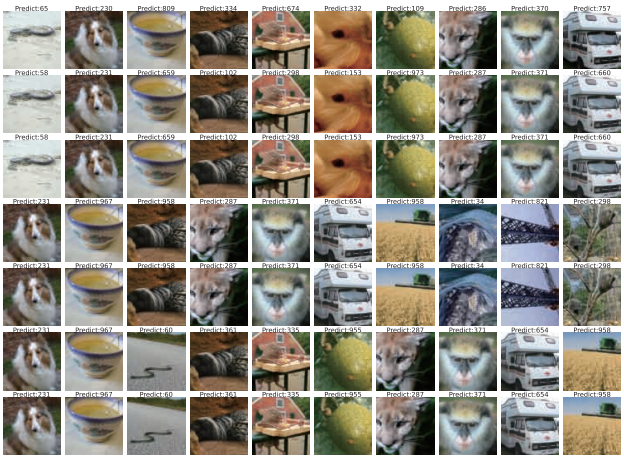

Figure 7: ℓ_1 adversarial examples generated on ImageNet

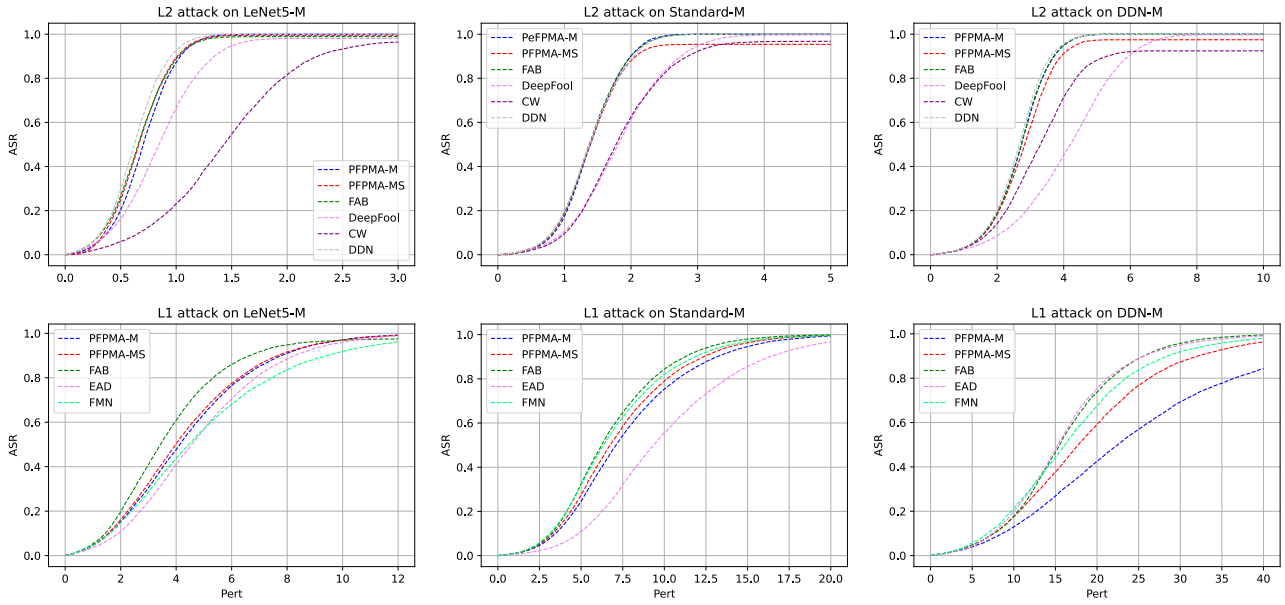

Figure 8: ASR_Perturbation curves of MNIST models

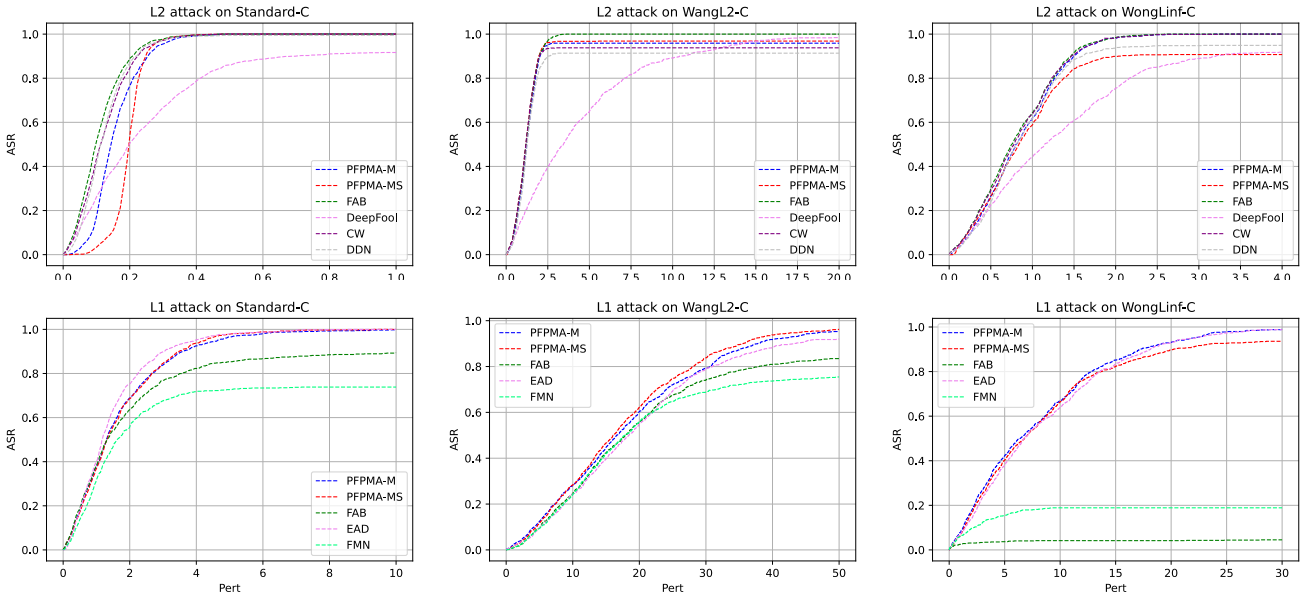

Figure 9: ASR_Perturbation curves of CIFAR10 models

existing methods. We believe our method could be used to evaluate and improve the reliability and robustness of deep learning systems applied in engineering.

REFERENCES

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *CoRR*, vol. abs/1412.6572, 2014.

[3] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, "Adversarial examples are not bugs, they are features," *Advances in neural information processing systems*, vol. 32, 2019.

[4] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, pp. 39–57, Ieee, 2017.

[5] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J.

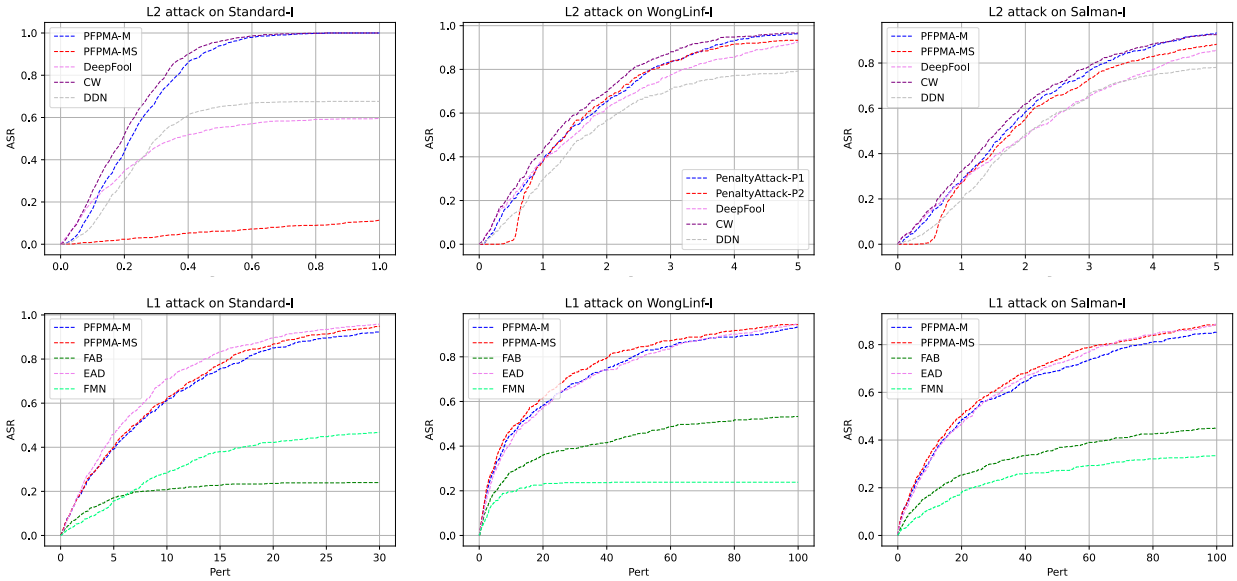

Figure 10: ASR_Perturbation curves of ImageNet models

Hsieh, "Ead: elastic-net attacks to deep neural networks via adversarial examples," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[6] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[7] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[8] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9185–9193, 2018.

[9] J. Rony, L. G. Hafemann, L. S. Oliveira, I. B. Ayed, R. Sabourin, and E. Granger, "Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4322–4330, 2019.

[10] M. Pintor, F. Roli, W. Brendel, and B. Biggio, "Fast minimum-norm adversarial attacks through adaptive norm constraints," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20052–20062, 2021.

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

[12] F. Croce and M. Hein, "Minimally distorted adversarial examples with a fast adaptive boundary attack," in *International Conference on Machine Learning*, pp. 2196–2205, PMLR, 2020.

[13] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear programming: theory and algorithms*. John wiley & sons, 2013.

[14] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, p. 2, 2012.

[15] A. Beck, *First-order methods in optimization*. SIAM, 2017.

[16] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[18] F. Croce, M. Andriushchenko, V. Sehwag, E. Debenedetti, N. Flammarion, M. Chiang, P. Mittal, and M. Hein, "Robustbench: a standardized adversarial robustness benchmark," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[19] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training," *arXiv preprint arXiv:2001.03994*, 2020.

[20] Z. Wang, T. Pang, C. Du, M. Lin, W. Liu, and S. Yan, "Better diffusion models further improve adversarial training," 2023.

[21] H. Salman, A. Ilyas, L. Engstrom, A. Kapoor, and A. Madry, "Do adversarially robust imagenet models transfer better?," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3533–3545, 2020.

[22] H. Kim, "Torchattacks: A pytorch repository for adversarial attacks," *arXiv preprint arXiv:2010.01950*, 2020.

[23] J. Rony and I. Ben Ayed, "Adversarial Library."