

# Method for Improving Quality of Adversarial Examples

Duc-Anh Nguyen<sup>1</sup><sup>a</sup>, Kha Do Minh<sup>1</sup><sup>b</sup>, Duc-Anh Pham<sup>2</sup><sup>c</sup> and Pham Ngoc Hung<sup>1</sup><sup>d</sup>

<sup>1</sup>VNU University of Engineering and Technology (VNU-UET),

Building E3, 144 Xuan Thuy Road, Cau Giay District, Hanoi, Vietnam

<sup>2</sup>University of Transport Technology, 54 Trieu Khuc Road, Thanh Xuan District, Hanoi, Vietnam

**Keywords:** Adversarial Example Generation, Deep Neural Network, Robustness, Autoencoder.


**Abstract:** To evaluate the robustness of DNNs, most of the adversarial methods such as FGSM, box-constrained L-BFGS, and ATN generate adversarial examples with small  $L_p$ -norm. However, these adversarial examples might contain many redundant perturbations. Removing these perturbations increases the quality of adversarial examples. Therefore, this paper proposes a method to improve the quality of adversarial examples by recognizing and then removing such perturbations. The proposed method includes two phases namely *the autoencoder training phase* and *the improvement phase*. In the autoencoder training phase, the proposed method trains an autoencoder that learns how to recognize redundant perturbations. In the second phase, the proposed method uses the trained autoencoder in combination with the greedy improvement step to produce more high-quality adversarial examples. The experiments on MNIST and CIFAR-10 have shown that the proposed method could improve the quality of adversarial examples significantly. In terms of  $L_0$ -norm, the distance decreases by about 82%-95%. In terms of  $L_2$ -norm, the distance drops by around 56%-81%. Additionally, the proposed method has a low computational cost. This shows the potential ability of the proposed method in practice.


## 1 INTRODUCTION


Deep neural networks (DNNs) are widely used in image classification (Carlini and Wagner, 2016; Ciregan et al., 2012; Sultana et al., 2019). Recent research has shown that even DNNs achieve high accuracy on the training set, these models might suffer from poor robustness (Baluja and Fischer, 2017; Carlini and Wagner, 2016; Eniser et al., 2019; Goodfellow et al., 2015; Gopinath et al., 2019; Kurakin et al., 2016; Zhang et al., 2019). The robustness of DNNs measures the degree to which these DNNs work correctly in the presence of slight perturbations. Term *adversarial example* is used to call an original input image added slight perturbations. While the original input images must be recognized correctly by the attacked DNNs, these DNNs are not able to detect the label of adversarial examples properly. Due to the existence of adversarial examples, this poses a security concern on the robustness of DNNs.


Targeted attack is a well-known approach to generate adversarial examples. These adversarial examples could be used as evidence to demonstrate the robustness of DNNs. There are many methods in this approach such as FGSM (Goodfellow et al., 2015), BIS (Kurakin et al., 2016), ATN (Baluja and Fischer, 2017), Carnili-Wagner (Carlini and Wagner, 2016), box-constrained L-BFGS (Szegedy et al., 2014), etc. The general purpose of these methods is to generate adversarial examples close to the original input images and classified as a target label. The target label is different from the ground-truth label. To compute the distance between adversarial examples and original input images, the most common metric is  $L_p$ -norm metric. Some popular  $L_p$ -norm could be referred to  $L_0$ -norm (Carlini and Wagner, 2016; Gopinath et al., 2019),  $L_2$ -norm (Baluja and Fischer, 2017; Carlini and Wagner, 2016; Szegedy et al., 2014), and  $L_\infty$ -norm (Goodfellow et al., 2015; Kurakin et al., 2016).

Although these methods could generate adversarial examples with small  $L_p$ -norm, these methods are still confronted by the problem of redundant perturbation. Specifically, these methods suggest their own objective functions containing the objective functions of the attacked DNNs. The objective functions of

<sup>a</sup> <https://orcid.org/0000-0002-6337-6254>

<sup>b</sup> <https://orcid.org/0000-0002-1917-8314>

<sup>c</sup> <https://orcid.org/0000-0002-6017-6740>

<sup>d</sup> <https://orcid.org/0000-0002-5584-5823>

these methods are minimized as much as possible. The result of the minimization is a set of adversarial examples with small  $L_p$ -norm. However, due to the non-linear property of DNNs, minimizing their defined objective functions might struggle at the local minimum. It means that the quality of adversarial examples might not be optimal. In other words, adversarial examples still might contain redundant perturbations. A perturbation is redundant if removing it from an adversarial example does not change the label of this adversarial example. If such redundant perturbations are removed from adversarial examples, the value of their defined objective functions would be smaller. The  $L_p$ -norm between the original input images and adversarial examples could be reduced significantly. As the result, this makes the adversarial examples closer to the original input images.

To mitigate the above problem, this paper proposes a method that detects and removes redundant perturbations with a low computational cost. We focus on  $L_0$ -norm and  $L_2$ -norm metrics. The proposed method includes two main phases, namely *the autoencoder training phase* and *the improvement phase*. In the autoencoder training phase, the proposed method trains an autoencoder. The purpose of this autoencoder is to recognize redundantly adversarial features in adversarial examples. An adversarial feature is redundant if it contains redundant perturbation. Therefore, it should be restored to its original value. The original value is the value of the corresponding feature on the original input image. In the improvement phase, an adversarial example is fed into the trained autoencoder to generate the first version of the improved adversarial example. However, because this improved adversarial example might still contain redundant perturbations, the proposed method uses a greedy improvement step to remove these perturbations.

The experiments on the MNIST dataset (Lecun et al., 1998a) and the CIFAR-10 dataset (Krizhevsky, 2012) have shown that the proposed method could enhance the quality of adversarial examples considerably. The proposed method is compared with a greedy method. As a result, the introduced method could achieve the  $L_0$ -norm reduction rate of 82%-95% and the  $L_2$ -norm reduction rate of 56%-81%. Additionally, the proposed method could improve the quality of new adversarial examples with a low computational cost. It takes several seconds to improve the quality of about 1,000 adversarial examples.

The rest of this paper is organized as follows. Section 2 provides the background of this topic. Section 3 describes a motivating example of this research. Section 4 describes a greedy method to enhance the qual-

ity of adversarial examples. The overview of the proposed method is shown in Section 5. Next, Section 6 presents the experiments to demonstrate the advantages of the proposed method. Section 7 delivers the overview of the related research. Finally, the conclusion is described in Section 8.

## 2 BACKGROUND

In this section, the paper provides the background about the adversarial example generation topic. We first review the basic concept of DNNs and the targeted attack. We then describe the overview of the stacked convolutional autoencoder. Finally, this paper reviews well-known adversarial example generation methods using  $L_p$ -norm metric.

### 2.1 Deep Neural Network

A DNN  $\mathbf{M}$  is made of  $h$  successive layers. A simple representation of a DNN is as follows:

$$\mathbf{M}(\mathbf{x}) = Q_{h-1}(Q_{h-2}(\dots(Q_0(\mathbf{x})))) \in \mathbb{R}^k \quad (1)$$

where  $\mathbf{x} \in \mathbb{R}^d$  is an input image of the DNN classifier,  $d$  is the number of features,  $k$  is the number of classes,  $h$  is the number of layers, and  $Q_i$  is the  $i$ -th layer. The input layer and the output layer are  $Q_0$  and  $Q_{h-1}$ , respectively. Each layer has one activation function. Some popular activation functions are ReLU and softmax. The output of the DNN  $\mathbf{M}$  is a probability vector of size  $k$ . The predicted label of an input  $\mathbf{x}$  is computed by  $\text{argmax} \mathbf{M}(\mathbf{x})$ .

### 2.2 Targeted Attack

The targeted attack is a popular approach to generate adversarial examples. Given an input image  $\mathbf{x}$  and a target label  $y^*$ , the purpose of the targeted attack is modifying this input to generate an adversarial example  $\mathbf{x}'$ . This adversarial example is classified as  $y^*$ , which is different from the ground-truth label of  $\mathbf{x}$ . Additionally,  $\mathbf{x}'$  should be close to  $\mathbf{x}$  as much as possible.

**Definition 1** (Adversarial Feature). *Given an input image  $\mathbf{x}$  and its corresponding adversarial example  $\mathbf{x}'$ , a feature  $x'_i$  on  $\mathbf{x}'$  is considered as adversarial feature if and only if  $x'_i \neq x_i$ , where  $x_i$  is the original value of  $x'_i$ .*

**Definition 2** (Redundant Perturbation). *Consider an adversarial feature, its perturbation  $\beta$  is redundant if and only if  $\mathbf{x}'$  and  $(\mathbf{x}' - \beta)$  are labelled the same. In this case, this feature is called a redundantly adversarial feature.*

**Definition 3** (Improved Adversarial Example). An improved adversarial example  $\mathbf{x}'_{im}$  is the result of removing redundant perturbations from its corresponding adversarial example  $\mathbf{x}'$ , in which  $\mathbf{x}'_{im}$  and  $\mathbf{x}'$  are assigned to the same label by  $\mathbf{M}$ .

**$L_p$ -norm.** To measure the distance between original input images and adversarial examples, the commonly used metric is  $L_p$ -norm. Given an input image  $\mathbf{x}$ , its corresponding adversarial example  $\mathbf{x}'$  should be identical to  $\mathbf{x}$  under  $L_p$ -norm metric as much as possible. The main reason is that it does not make sense if an adversarial example is too different from the original input image. The  $L_p$ -norm distance between the input image  $\mathbf{x}$  and its adversarial example  $\mathbf{x}'$  is formally defined as follows:

$$L_p(\mathbf{x}, \mathbf{x}') = \sqrt[p]{\sum_i (x_i - x'_i)^p} \quad (2)$$

### 2.3 Stacked Convolutional Autoencoder

A stacked convolutional autoencoder considers the structure of 2D and 3D images. This type of autoencoder includes one encoder and one decoder. In the encoder part, a layer can be down-sampling, convolutional, and fully-connected. In the decoder part, a layer can be up-sampling, deconvolutional, and fully-connected. The typical objective function of the stacked convolutional autoencoder is as follows:

$$\sum_{\mathbf{x}_{in}} L_2^2(\mathbf{x}_{in}, \mathbf{x}_{out}) \quad (3)$$

where  $\mathbf{x}_{in}$  is the input image on the training set,  $\mathbf{x}_{out}$  is the corresponding reconstructed image of  $\mathbf{x}_{in}$ . The distance between  $\mathbf{x}_{in}$  and  $\mathbf{x}_{out}$  is computed by using metric  $L_2$ -norm.

### 2.4 Popular Targeted $L_p$ -norm Attack

This part describes three common targeted attacks using  $L_p$ -norm metric. While FGSM uses  $L_\infty$ -norm, box-constrained L-BFGS and ATN use  $L_2$ -norm. To generate an adversarial example, these methods could add perturbations to all features on an input image.

#### 2.4.1 Box-constrained L-BFGS

Szegedy et al. (2014) proposed box-constrained L-BFGS to generate adversarial examples. Their method solves the following box-constrained optimization problem:

$$\begin{aligned} & \text{minimize } (1-c) \cdot L_2^2(\mathbf{x}', \mathbf{x}) + c \cdot f(y^*, \mathbf{M}(\mathbf{x}')) \\ & \text{such that } \mathbf{x}' \in [0, 1]^d \end{aligned} \quad (4)$$

where  $f$  is a function to compute the difference between  $\mathbf{M}(\mathbf{x}')$  and the target label  $y^*$ , and  $c$  is the weight to balance two terms. A common choice of  $f$  is cross-entropy.

#### 2.4.2 FGSM

Goodfellow et al. (2015) proposed a fast gradient sign method (FGSM) to generate adversarial examples. Specifically, given an input image  $\mathbf{x}$ , and a value  $\varepsilon$ , its corresponding adversarial example  $\mathbf{x}'$  is computed as follows:

$$\mathbf{x}' = \mathbf{x} - \varepsilon \cdot \text{sign}(\nabla_{\mathbf{x}} f(\mathbf{x}, y^*, \zeta)) \quad (5)$$

where  $\text{sign}(\cdot)$  returns the sign of  $\nabla_{\mathbf{x}} f(\mathbf{x}, y^*, \zeta)$  and  $\varepsilon$  is a positive value to shift values of all features in  $\mathbf{x}$ .

#### 2.4.3 ATN

Baluja and Fischer (2017) proposed ATN to generate adversarial examples based on an autoencoder. The input of the autoencoder is an input image  $\mathbf{x}$ . The output is an adversarial example  $\mathbf{x}'$ . The authors suggested using  $L_2$ -norm to compute the distance between  $\mathbf{x}'$  and  $\mathbf{x}$ . The objective function of sample  $\mathbf{x}$  is as follows:

$$(1 - \phi) \cdot L_2(\mathbf{x}, \mathbf{x}') + \phi \cdot L_2(\mathbf{M}(\mathbf{x}'), r_\alpha(\mathbf{M}(\mathbf{x}), y^*)) \quad (6)$$

where  $r_\alpha(\cdot)$  is a reranking function,  $\phi$  is the weight between the two terms of the objective function,  $L_2(\mathbf{M}(\mathbf{x}'), r_\alpha(\mathbf{M}(\mathbf{x}), y^*))$  is the  $L_2$ -norm distance between  $\mathbf{M}(\mathbf{x}')$  and the expected probability vector  $r_\alpha(\mathbf{M}(\mathbf{x}), y^*)$ .

The reranking function  $r_\alpha(\cdot)$  is defined as follows:

$$\begin{aligned} & r_\alpha(\mathbf{M}(\mathbf{x}), y^*) \\ & = \text{norm} \left( \left\{ \begin{array}{l} \alpha \cdot \max(\mathbf{M}(\mathbf{x})) \text{ if } i = y^* \\ \mathbf{M}(\mathbf{x})_i \text{ otherwise} \end{array} \right\}_{i \in \{0..k-1\}} \right) \end{aligned} \quad (7)$$

where  $\alpha > 1$  is used to specify how much larger  $y^*$  should be than  $\max(\mathbf{M}(\mathbf{x}))$ . Function  $\text{norm}(\cdot)$  normalizes its input to a probability distribution.

## 3 MOTIVATING EXAMPLE

Many methods are proposed to generate adversarial examples with small  $L_p$ -norm. For  $L_0$ -norm, some well-known methods could be referred to Carnili-Wagner  $L_0$  (Carlini and Wagner, 2016) and DeepCheck (Gopinath et al., 2019). Concerning  $L_2$ -norm, Carnili-Wagner  $L_2$  (Carlini and Wagner, 2016), box-constrained L-BFGS (Szegedy et al., 2014), and

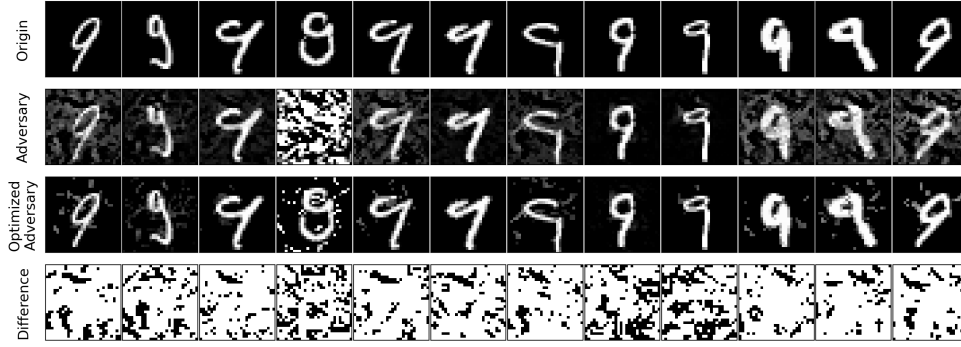


Figure 1: A motivating example.

ATN (Baluja and Fischer, 2017) are common methods. We observed that the quality of the adversarial example generated by these methods could be enhanced more in terms of  $L_0$ -norm and  $L_2$ -norm.

Figure 1 shows a motivating example of this research. The first row *Origin* shows the original images which are modified to produce adversarial examples. The second row *Adversary* contains adversarial examples generated by applying box-constrained L-BFGS. The third row shows the optimized adversary. The last row represents the difference between the second row and the third row. White pixel denotes the difference and dark pixel is corresponding to identicalness.

As can be seen, by comparing the second row and the third row, some optimized adversarial examples are very different from the corresponding adversarial examples. In other words, the adversarial examples in the second row still contain many redundant perturbations. To minimize  $L_0$ -norm and  $L_2$ -norm, these redundant perturbations should be recognized and removed from the adversarial examples. Therefore, this paper proposes a method to enhance the quality of adversarial examples generated by any adversarial attacks with low computational cost. We focus on  $L_0$ -norm and  $L_2$ -norm.

## 4 GREEDY METHOD

To the best of our knowledge, there is no similar work to improve the quality of adversarial examples generated by adversarial attacks. Most of the research on this topic aims to generate high-quality adversarial examples during the adversarial attack, not after the adversarial attack. Therefore, to show the effectiveness of the proposed method, this research compares it with a greedy approach. We also explain why although the greedy method could produce good results of adversary improvement, it has several prob-

lems when applying in practice.

Algorithm 1 describes the overview of the greedy method to enhance the quality of an adversarial example. The input includes an adversarial examples  $\mathbf{x}'$ , its corresponding input image  $\mathbf{x}$ , an attacked model  $\mathbf{M}$ , and a step  $\alpha$ . The target label is the predicted label of  $\mathbf{x}'$ . The output is an improved adversarial example. The greedy method is used as a baseline in the experiments.

---

Algorithm 1: Greedy Method: Improve the quality of adversarial examples in terms of  $L_0$ -norm and  $L_2$ -norm.

---

**Input:** adversarial example  $\mathbf{x}'$ , input image  $\mathbf{x}$ , DNN  $\mathbf{M}$ , step  $\alpha$

**Output:** An improved adversarial example

```

1:  $l = \operatorname{argmax}(\mathbf{M}(\mathbf{x}'))$ 
2: while  $\alpha > 0$  do
3:    $F_{adv} \leftarrow \text{GET-ADVERSARIAL-FEATURES}(\mathbf{x}, \mathbf{x}')$ 
4:    $B_{adv} = F_{0..\alpha-1} \cup F_{\alpha..2*\alpha-1} \cup \dots \cup F_{n*\alpha..||F_{adv}||}$ 
5:   for  $F_{i..j} \in B_{adv}$  do
6:      $\mathbf{x}' = \text{Update } x'_i \leftarrow x_i, \dots, x'_j \leftarrow x_j$ 
7:     if  $\operatorname{argmax}(\mathbf{M}(\mathbf{x}')) \neq l$  then
8:       Undo the step 6
9:     end if
10:  end for
11:   $\alpha \leftarrow \lfloor \alpha/2 \rfloor$ 
12: end while
13: return  $\mathbf{x}'$ 

```

---

In the greedy method, a set of adversarial features (denoted by  $F_{adv}$ ) is detected by comparing  $\mathbf{x}'$  and  $\mathbf{x}$  (line 3). The order of adversarial features is arranged from the top left corner to the bottom right corner of  $\mathbf{x}'$ . The greedy method attempts to restore the adversarial features on this set to the original values. The original value of adversarial feature  $x'_i$  is  $x_i$ . We use a step (denoted by  $\alpha \geq 1$ ) to determine the number of adversarial features restored at the same time.

The greedy method specifies subsets of adversarial features by dividing the set  $F_{adv}$  (line 4). Subset  $F_{i..j}$  stores the adversarial features from  $i^{\text{th}}$  position



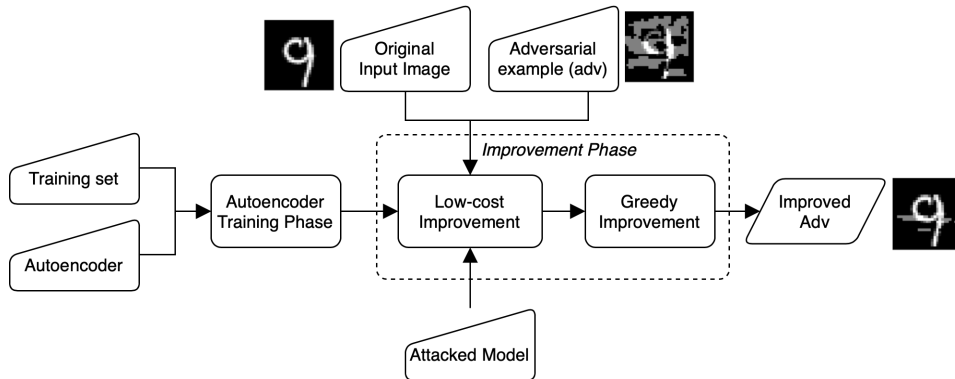


Figure 2: The overview of the proposed method.

to  $j^{\text{th}}$  position on  $F_{adv}$ . After that, we iterate over these subsets. At each iteration, the adversarial features from  $i^{\text{th}}$  position to  $j^{\text{th}}$  position on  $\mathbf{x}'$  attempts to be restored to the original values (line 6). There are two cases. If this restoration changes the label of  $\mathbf{x}'$  (line 7 - 8), the modification on  $\mathbf{x}'$  is reverted to preserve its predicted label. Otherwise, the modification on  $\mathbf{x}'$  is kept. The algorithm moves to the next subset of  $F_{adv}$ .

When more adversarial features are restored at the same time, the probability to change the decision of the attacked model on  $\mathbf{x}'$  is larger. Therefore, after iterating all subsets  $F_{i..j}$ , the step  $\alpha$  decreases by twice (line 11). The algorithm performs the restoration of adversarial features with a smaller size of subset  $F_{i..j}$ . The algorithm terminates when  $\alpha$  is not greater than 1.

However, the greedy method has two main limitations. Firstly, the greedy method does not support the generalization ability. The experiences of improving an adversarial example in the past are not utilized to enhance new adversarial examples. Secondly, this method has poor performance. The total cost of executing line 7 could be time-consuming. The DNN model predicts the label of the temporary modification of an adversarial example. It requires the initialization of DNN, many computations on the DNN graph to obtain a result. Especially, this cost could be huge when the number of redundantly adversarial features is large. For example, in FGSM, most of the features on an input image would be modified to generate an adversarial example. It means that more number iterations (i.e., line 5) would be performed.

## 5 THE PROPOSED METHOD

This paper proposes a method to mitigate the limitation of the greedy method. The overview of the proposed method is illustrated in Fig. 2. The proposed

method includes two main phases, namely *autoencoder training phase* and *improvement phase*. The purpose of the autoencoder training phase is to train an autoencoder, which is an input of the second phase. The purpose of the improvement phase is to enhance the quality of adversarial examples in terms of  $L_0$ -norm and  $L_2$ -norm distance.

### 5.1 Autoencoder Training Phase

The input of this phase includes the training set and the architecture of an autoencoder denoted by  $g$ . The output of this phase is a trained autoencoder.

The training set is a set of  $(\mathbf{x}', s_{\mathbf{x}'})$ , where  $\mathbf{x}' \in \mathbb{R}^d$  is an unimproved adversarial example and  $s_{\mathbf{x}'} \in \mathbb{R}^d$  is a 0-1 vector. Each value  $s_{\mathbf{x}'}[i]$  represents the probability to restore the adversarial feature  $\mathbf{x}'[i]$  to the original values:

$$s_{\mathbf{x}'}[i] = \begin{cases} 1 & \text{if } x'_i \text{ is redundant} \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

For example, consider an unimproved adversarial example  $\mathbf{x}' = [0, 0, 1, \dots, 0.5, 0]^T \in \mathbb{R}^{784}$ . The corresponding input image of this adversarial example is  $\mathbf{x} = [1, 0, 1, \dots, 0.5, 0]^T \in \mathbb{R}^{784}$ . Restoring the adversarial feature  $x'_0$  (i.e., value 0) to the original value (i.e., value 1) produces a modified adversarial example. If the label of the modified adversarial example is the same as that of  $\mathbf{x}'$ ,  $x'_0$  is redundant. Therefore,  $s_{\mathbf{x}'}[0]$  is assigned to 1. Otherwise,  $s_{\mathbf{x}'}[0]$  is assigned to 0.

The architecture of the autoencoder  $g$  is defined manually. The output of the autoencoder from an adversarial example  $\mathbf{x}'$  is denoted by  $g(\mathbf{x}')$ . For an adversarial example  $\mathbf{x}'$  on the training set, the objective of the autoencoder is defined as follows:

$$\frac{1}{|\mathbf{x}'|} \sum_{i=0}^{|\mathbf{x}'|-1} f(s_{\mathbf{x}'}[i], g(\mathbf{x}')[i]) \quad (9)$$

where  $f$  is binary cross-entropy.

The proposed autoencoder in Equation 9 is different from the traditional autoencoder. The main difference is the output  $g(\mathbf{x}')$  of the autoencoder. In the traditional autoencoder,  $g(\mathbf{x}')$  should be identical to the input  $\mathbf{x}'$ . In this case, the metric *mean squared error* is usually used to measure the distance between  $\mathbf{x}'$  and  $g(\mathbf{x}')$ . However, in our proposed autoencoder,  $g(\mathbf{x}')$  is a probability vector. Therefore, we use binary cross-entropy to compare  $g(\mathbf{x}')$  and  $s_{\mathbf{x}'}$ .

## 5.2 Improvement Phase

This phase improves the quality of adversarial examples in terms of  $L_0$ -norm and  $L_2$ -norm. The input of this phase is an attacked model  $M$ , a trained autoencoder  $g$  (i.e., the result of the previous phase), an unimproved adversarial example  $\mathbf{x}'$ , and its corresponding input image  $\mathbf{x}$ . The output of this phase is an improved adversarial example. This phase has two main steps including *a low-cost improvement step* and *a greedy improvement step*.

**Low-cost Improvement Step.** In this step, the trained autoencoder  $g$  is used to generate the first version of the improved adversarial example, denoted by  $\mathbf{x}'_1$ . Because  $g(\mathbf{x}')$  is a probability vector, we need a threshold  $\delta \in [0, 1]$  to determine whether  $g(\mathbf{x}') [i]$  is classified as *redundant* or not. The first version  $\mathbf{x}'_1$  is created as follows:

$$\mathbf{x}'_1 [i] = \begin{cases} x'_i & \text{if } g(\mathbf{x}') [i] \leq \delta \\ x_i & \text{otherwise} \end{cases} \quad (10)$$

The first version  $\mathbf{x}'_1$  is then predicted by the attacked model  $M$ . If the label of  $\mathbf{x}'_1$  is the same as that of  $\mathbf{x}'$ ,  $\mathbf{x}'_1$  is then fed into the second step of this phase. Otherwise, the value of threshold  $\delta$  should increase. When  $\delta$  increases, there is fewer number of redundantly adversarial features. As a result,  $\mathbf{x}'_1$  is more close to  $\mathbf{x}'$ . The attacked model  $M$  tends to predict the label of  $\mathbf{x}'_1$  and  $\mathbf{x}'$  the same.

The low-cost improvement step takes low computational cost. The experiment has shown that this autoencoder takes around 1-2 seconds for the MNIST dataset (Lecun et al., 1998a) and around 3-4 seconds for the CIFAR-10 dataset (Krizhevsky, 2012) for about 1,000 adversarial examples.

**Greedy Improvement Step.** It is observed that the first version  $\mathbf{x}'_1$  could be enhanced more. There are some adversarial features on  $\mathbf{x}'_1$  that could be restored to the original values. Therefore, the proposed method applies the greedy method described in Sect. 4 to find out such adversarial features. The output of this step is the second version of the improved adversarial example, denoted by  $\mathbf{x}'_2$ . The proposed

method returns  $\mathbf{x}'_2$  as the final improved adversarial example of  $\mathbf{x}'$  and terminates.

## 6 EXPERIMENTAL RESULTS

To demonstrate the advantages of the proposed method, the experiments address the following questions:

- **RQ1 - Success Rate of Autoencoder:** Does the trained autoencoder improve most of the adversarial examples?
- **RQ2 - Quality:** Does the proposed improvement phase reduce the  $L_0$ -norm and  $L_2$ -norm of the unimproved adversarial examples?
- **RQ3 - Performance:** Does the proposed improvement phase achieve good performance when dealing with the unimproved adversarial examples?

For the autoencoder training phase, we answer *RQ1* to investigate the effectiveness of the trained autoencoders. For the improvement phase, the experiments answer *RQ2* and *RQ3* by comparing this phase with the greedy method. The experiments are performed on Google Colab<sup>1</sup>.

### 6.1 Configuration

#### 6.1.1 Dataset

The experiments are conducted on the MNIST dataset (Lecun et al., 1998a) and the CIFAR-10 (Krizhevsky, 2012) dataset. These datasets are used commonly to evaluate the robustness of DNN models (Zhang et al., 2019). About the MNIST dataset, it is a collection of handwritten digits. The training set has 60,000 images and the test set contains 10,000 images of size  $28 \times 28 \times 1$ . There are 10 labels representing the digits from 0 to 9. Concerning the CIFAR-10 dataset, it is a collection of images labelled as aeroplanes, trucks, birds, cats, etc. It has 50,000 images on the training set and 10,000 images of size  $32 \times 32 \times 3$  on the test set. One pixel is described in a combination of red, green, and blue channels.

#### 6.1.2 Attacked Model

For the MNIST dataset, the experiments use LeNet-5 (Lecun et al., 1998b). This architecture is introduced to recognize handwritten digits. For the CIFAR-10 dataset, the experiments use

<sup>1</sup><https://colab.research.google.com/>

AlexNet (Krizhevsky et al., 2017). Compared to the architecture of LeNet-5, the architecture of AlexNet is more complicated. This model is used for general-purpose image classification. The accuracies of the trained models are shown in Table 1.

Table 1: The accuracy of attacked models.

Dataset	Model	Training set	Test set
MNIST	LeNet-5	99.86%	98.82%
CIFAR-10	AlexNet	99.70%	76.22%

### 6.1.3 Adversarial Example

The adversarial examples are generated by applying three adversarial attacks including FGSM, box-constrained L-BFGS, and ATN. The purpose is to demonstrate the effectiveness of the proposed method when dealing with various adversarial examples. For FGSM and L-BFGS, the experiments use their original equations. For ATN, we make several small modifications in Equation 6 to generate more high-quality adversarial examples. We empirically use cross-entropy instead of  $L_2$ -norm to compare the distance between the expected probability and the predicted probability. Additionally, we use the identity function instead of the reranking function.

If the original input images labelled  $m$  are modified into an adversarial example classified as  $n$ , the experiments denote as the attack  $m \rightarrow n$  for simplicity. For the MNIST dataset, the attack is  $9 \rightarrow 4$ . For the CIFAR-10 dataset, the attack is *truck*  $\rightarrow$  *horse*.

### 6.1.4 The Proposed Method

**Autoencoder Training Phase.** In the first phase of the proposed method, we train autoencoders to recognize redundant perturbations existing inside an adversarial example. Each autoencoder is trained on 6,000 samples by using the Equation 9. A sample is a set of  $(\mathbf{x}', s_{\mathbf{x}'})$ , where  $\mathbf{x}'$  is unimproved adversarial examples. These training sets are denoted by  $\mathbf{X}_{train}$ . These autoencoders are trained in 300 epochs.

The architectures of autoencoders should be defined based on the experience of machine learning testers. The experiments use stacked convolutional autoencoders because these autoencoders could learn the characteristics of 2D and 3D images better than traditional autoencoders. Table 2 describes the general architecture of the autoencoders used in the proposed method. The architectures of these autoencoders are the same, except for the input layer and the output layer. The layer before the output layer uses sigmoid to compute the probability for binary classification.

Table 2: The architectures of the autoencoders used in the proposed method. For MNIST, the shape of input is (28, 28, 1). For CIFAR-10, the shape of input is (32, 32, 3).

Layer Type	Configuration
<b>Input</b>	#width $\times$ #height $\times$ #channel
Conv2D + ReLU	kernel = (3, 3), #filters = 64
Conv2D + ReLU	kernel = (3, 3), #filters = 64
BatchNormalization	
MaxPooling	pool' size = (2, 2)
Conv2D + ReLU	kernel = (3, 3), #filters = 32
Conv2D + ReLU	kernel = (3, 3), #filters = 32
BatchNormalization	
MaxPooling2D	pool' size = (2, 2)
Conv2D + ReLU	kernel = (3, 3), #filters = 32
Conv2D + ReLU	kernel = (3, 3), #filters = 32
BatchNormalization	
UpSampling2D	pool' size = (2, 2)
Conv2D + ReLU	kernel = (3, 3), #filters = 64
Conv2D + ReLU	kernel = (3, 3), #filters = 64
BatchNormalization	
UpSampling2D	pool' size = (2, 2)
Conv2D + Sigmoid	kernel = (3, 3), #filters = #channel
<b>Output</b>	#width $\times$ #height $\times$ #channel

**Improvement Phase.** The experiments use 1,000 adversarial examples denoted by  $\mathbf{X}_{rest}$  to evaluate the effectiveness of the proposed method. These adversarial examples are not used to train autoencoders. The configuration of improvement is as follows:

- **Low-cost Improvement Step.** Unimproved adversarial examples are fed into the trained autoencoders.
- **Greedy Improvement Step.** For MNIST, the value of step  $\alpha$  (in Algorithm 2) is 6. While the number of features on CIFAR-10 is much larger than that of MNIST, the initial value of step  $\alpha$  for CIFAR-10 should be larger. For CIFAR-10, we empirically assign the initial value of step  $\alpha$  to 60.

### 6.1.5 Baseline

The proposed method is compared with the greedy method. It is the baseline in the experiment. The input of the baseline is 1,000 unimproved adversarial examples. These adversarial examples are the same as the adversarial examples used in the improvement phase of the proposed method.

Additionally, the configuration  $\alpha$  of the baseline is the same as the greedy improvement step in the proposed method. Specifically, for MNIST, the initial value of step  $\alpha$  is 6. For CIFAR-10, the initial value of step  $\alpha$  is 60.

## 6.2 Answer for RQ1 - Success Rate of Autoencoder

This section investigates the success rate of the autoencoders. These autoencoders are trained on  $\mathbf{X}_{train}$  consisting of around 6,000 samples and evaluated on  $\mathbf{X}_{test}$  consisting of about 1,000 samples. The success rate is the percentage of the unimproved adversarial examples which are enhanced successfully by the autoencoders. If the success rate is close to 100%, most unimproved adversarial examples are enhanced their quality by the trained autoencoders.

We observed that increasing threshold  $\delta$  in Equation 10 would increase the success rate. The main reason is that when the value of  $\delta$  goes up, there is less number of adversarial features restored to the original values. In other words, the modified adversarial example is closer to the unimproved adversarial example. As a result, the label of the modified adversarial example tends to be the same as that of the unimproved adversarial example. For example, Fig. 3 describes the impact of threshold on the success rate. As can be seen, the success rate increases to nearly 100% when  $\delta$  goes up to nearly 1.

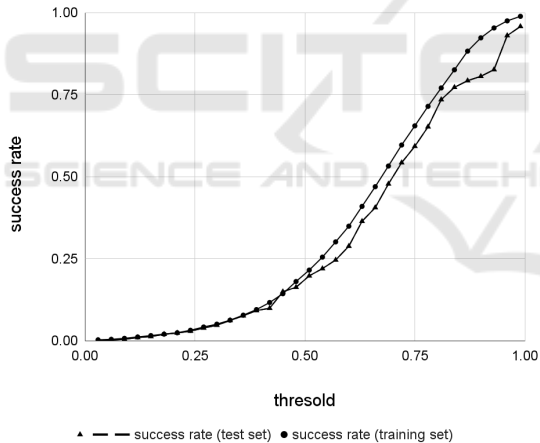


Figure 3: Trend of success rate when using different thresholds. The adversarial examples are generated by FGSM. The attacked model is LeNet-5 trained on MNIST.

The experiments continue investigating more deeply the impact of three values of  $\delta$  including 0.93, 0.96, and 0.99 on success rate. The main reason of using these  $\delta$  is that it would result in high success rate. We compute the average success rate rather than reporting the success rate of each threshold individually. Table 3 shows the average success rate of the trained autoencoders. For example, 97.2% means the trained autoencoders could improve the quality of about  $|\mathbf{X}_{train}| \times 97.2\%$  adversarial examples. As can be seen, the trained autoencoders could enhance the

quality of most adversarial examples on  $\mathbf{X}_{train}$ , from around 87% to about 97%. For  $\mathbf{X}_{test}$ , the success rates are from around 81% to about 90%. This shows that the trained autoencoders can recognize redundant perturbation inside adversarial examples effectively.

Table 3: Average success rate of the trained autoencoders.

Dataset	Method	Training set	Test set
MNIST	FGSM	97.2%	90.4%
	L-BFGS	89.3%	87.1%
	ATN	95.2%	92.2%
CIFAR-10	FGSM	87.4%	81.3%
	L-BFGS	87.3%	83.3%
	ATN	87.7%	86.4%

The trained autoencoders in RQ1 are used to answer RQ2 and RQ3. These two research questions only evaluate the effectiveness of the improvement phase. Given the same subset of unimproved adversarial examples, the experiments investigate if the proposed improvement phase works better than the greedy method. Two compared metrics are the reduction rate of  $L_0/L_2$ -norm and the performance.

## 6.3 Answer for RQ2 - Quality Improvement

This section demonstrates how the proposed improvement phase enhances the quality of  $\mathbf{X}_{test}$  in terms of  $L_0$ -norm and  $L_2$ -norm. These adversarial examples are not used to train the autoencoder. The compared method is the greedy method. The compared metric is the reduction rate, which is computed by  $(a - b)/a \in [0, 1)$ , where  $a$  is the distance before the improvement,  $b$  is the distance after the improvement. It is expected that  $b$  should be as small as possible.

In the proposed improvement phase, the process of improving adversarial examples is as follows. In the first step,  $\mathbf{X}_{test}$  is fed into the low-cost improvement step. The improved adversarial examples are the first version of the improvement. In the second step, the output of the first step is fed into the greedy improvement step to enhance its quality more.

In the low-cost improvement step, the parameter  $\delta$  of the trained autoencoders should be chosen to improve the quality of  $\mathbf{X}_{test}$  as much as possible. We empirically observed that  $\delta \in [0.8, 1)$  would result in good reduction rate of  $L_0$ -norm and  $L_2$ -norm. For example, Fig. 4 describes the impact of threshold  $\delta$  on the reduction rate of  $L_0$ -norm and  $L_2$ -norm. As can be seen, increasing the threshold  $\delta$  to around 0.9x improves the reduction rate of both  $L_0$ -norm and  $L_2$ -norm significantly. The reduction rates then decrease slightly when  $\delta$  is very close to 1.

Based on the above observation, the experiments



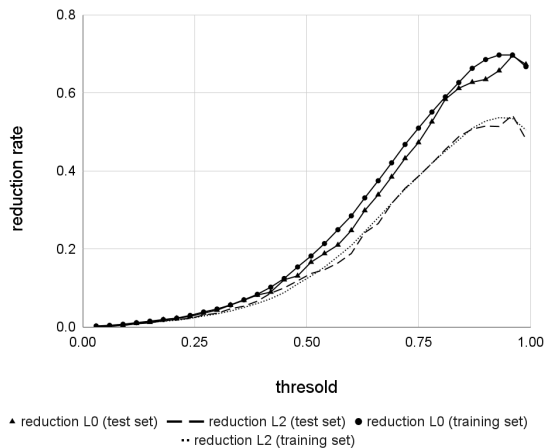


Figure 4: Trend of reduction rate when using different thresholds  $\delta$ . The adversarial examples are generated by FGSM. The attacked model is LeNet-5 trained on MNIST.

choose  $\delta$  used in RQ1 (i.e., 0.93, 0.96, and 0.99) to make comparison with the greedy method. Table 4 shows the average reduction rate of  $L_0$ -norm and  $L_2$ -norm. Generally, the proposed improvement phase produces better results than the greedy method in 8/12 cases. The experiment shows that the proposed improvement phase could achieve the reduction rate from around 82% to about 95% in terms of  $L_0$ -norm. Concerning  $L_2$ -norm, the reduction rate of the proposed improvement phase is from around 56% to approximately 81%.

Table 4: The average reduction rate (%) of  $L_0$ -norm and  $L_2$ -norm on  $\mathbf{X}_{test}$ . Better values are bold. Notion \* means that we ignore the process of training autoencoder.

Dataset	Method	$L_0$ -norm		$L_2$ -norm	
		Proposal*	Greedy	Proposal*	Greedy
MNIST	FGSM	<b>82.58</b>	82.40	<b>68.91</b>	68.07
	L-BFGS	<b>82.38</b>	82.30	68.01	<b>72.79</b>
	ATN	<b>95.20</b>	94.27	<b>59.67</b>	56.77
CIFAR-10	FGSM	<b>85.27</b>	84.92	<b>81.07</b>	79.31
	L-BFGS	83.90	<b>87.14</b>	63.8	<b>66.54</b>
	ATN	86.45	<b>88.69</b>	<b>68.27</b>	66.09

Figure 5 shows some examples of the improved adversarial examples by applying the proposed improvement phase. Row *origin* represents the input images, which are modified to generate adversarial examples. The result of adversarial attacks on these input images is stored in the second row *unimproved adv*. The adversarial examples in this row are not improved by applying the proposed improvement phase. The third row *low-cost adv* shows the first version of the improved adversarial examples. This row is the result of applying the low-cost improvement step. The last row *improved adv* shows the result of applying the greedy method. The first version of the

improved adversarial examples is fed into the greedy method to generate more high-quality results. As can be seen, in terms of human perception, the low-cost improvement step could enhance the quality of unimproved adversarial examples significantly. However, there are some redundantly adversarial perturbations in the third row. By applying the greedy improvement step, such perturbations are detected and removed.

## 6.4 Answer for RQ3 - Performance

This section evaluates the performance of the proposed improvement phase when dealing with  $\mathbf{X}_{test}$ . These adversarial examples are used in RQ2. This experiment uses different values of threshold  $\delta$  (i.e., 0.93, 0.96, and 0.99) as discussed in RQ1 and RQ2.

Table 5 shows the performance comparison in seconds between the proposed improvement phase and the greedy method. The computational cost of the proposed improvement phase includes *the low-cost improvement step* and *the greedy improvement step*. In the low-cost improvement step,  $\mathbf{X}_{test}$  are fed into the trained autoencoders to retrieve the first version of the improved adversarial examples. However, we observed that these improved adversarial examples could be enhanced more in terms of  $L_0$ -norm and  $L_2$ -norm. Therefore, these improved adversarial examples are put into the greedy improvement step to obtain the second version of improvement.

Table 5: The average performance comparison (seconds) between the proposed improvement phase and the greedy method on the test set. Better values are bold.

Dataset	Method	Proposed improvement phase			Greedy
		Low-cost	Greedy	$\Sigma$	
MNIST	FGSM	1.1	13.1	<b>14.2</b>	80.3
	L-BFGS	1.2	27.6	<b>28.8</b>	98.3
	ATN	1.4	20.1	<b>21.5</b>	120.2
CIFAR-10	FGSM	3.7	29.1	<b>32.8</b>	250.8
	L-BFGS	3.5	46.1	<b>49.6</b>	261.1
	ATN	3.4	62.1	<b>65.5</b>	254.5

As can be seen, the performance of the proposed improvement phase outperforms that of the greedy method. On the MNIST dataset, the proposed improvement phase usually requires from around 14.2 seconds to about 28.8 seconds, which is from roughly 3 times faster to around 6 times faster than that of the greedy method. On the CIFAR-10 dataset, the time of the proposed improvement phase is approximately 4 times to 8 times faster than that of the greedy method. The main reason is that in the proposed improvement phase, most of the adversarial features are restored to the original values by applying the trained autoencoders. As a result, there is only a small set of redundant features restored by the greedy improvement step. This helps to reduce the cost of the greedy im-

provement step, which is a time-consuming process.

Combining the result of research questions, the experiments show three insights. The first insight is that the proposed improvement phase could improve the quality of most adversarial examples. The second result is that the proposed improvement phase could improve the quality of new adversarial examples considerably. The third insight is that the proposed improvement phase has a low computational cost when dealing with new adversarial examples.

## 7 RELATED WORK

This section presents an overview of related research. Firstly, we discuss some well-known adversarial attacks using  $L_p$ -norm metric. Secondly, we clarify the novelty of the proposed method by discussing the state of adversarial example improvement methods.

**Adversarial Example Generation.** Many adversarial example generation methods are introduced to evaluate the robustness of DNNs. These methods add perturbations to an input image to generate an adversarial example. Several common methods are described as follows.

Box-constrained L-BFGS (Szegedy et al., 2014) minimizes  $L_2$ -norm by proposing an objective function for each input image. This objective ensures that the modification of an input image is small while it

is classified as a target label. However, this method does not treat the complex DNNs well because of the highly nonlinear property of the objective function.

Similarly to box-constrained L-BFGS, DeepFool (Moosavi-Dezfooli et al., 2015) aims to generate adversarial examples with small  $L_2$ -norm. This method assumes that the network to be completely linear. They compute a solution on the tangent plane (orthogonal projection) of a point on the classifier function. However, DNNs are not linear, this method then takes a step towards that solution, and repeats the process. The search terminates when an adversarial example is found.

FGSM (Goodfellow et al., 2015) adds perturbation to all features. This method has a low computational cost. However, the generated adversarial examples usually contain visible noises, which is not realistic in practice. Additionally, the  $L_\infty$ -norm distance is sensitive to the value of the input parameter.

Carnili-Wagner (Carlini and Wagner, 2016) improves the box-constrained L-BFGS by reducing the highly nonlinear property of the objective function. They apply the change of variable technique and use a simpler objective function. Their experiments have shown that the perturbation is much smaller than box-constrained L-BFGS.

Unlike the above methods, ATN (Baluja and Fischer, 2017) trains autoencoder for adversarial attack. The autoencoder could be used to generate adversar-

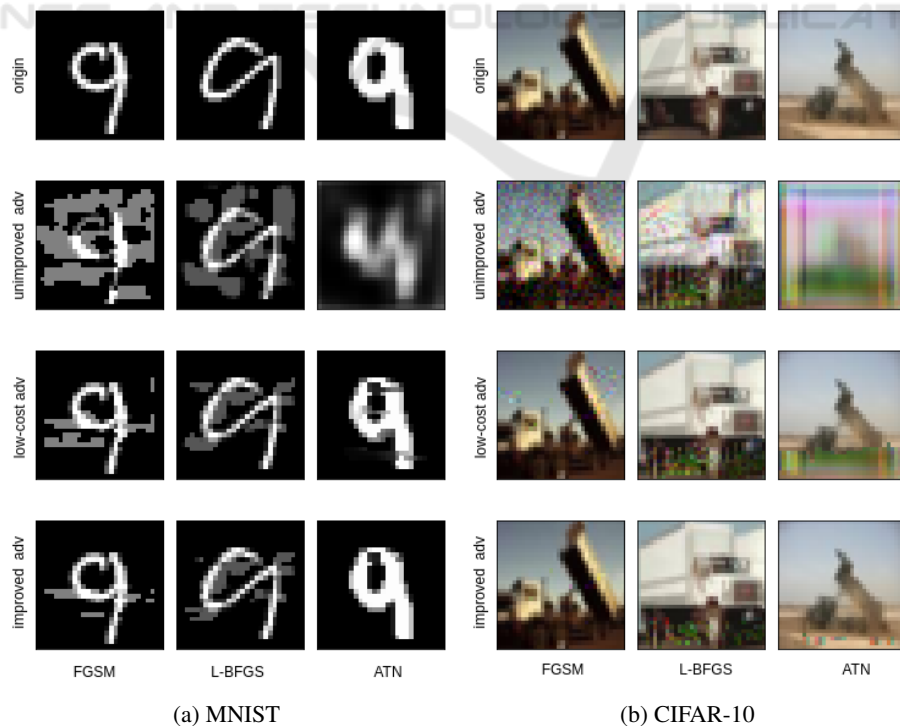


Figure 5: Examples of the improved adversarial examples by the proposed improvement phase.

ial examples from new input images with a low computational cost. Their method has the generalization ability, which is not supported by most of the existing methods.

DeepCheck (Gopinath et al., 2019) is the first symbolic execution tool for testing DNNs. This method translates a DNN into a Java program. From an input image, the program is instrumented and executed to get the execution path. After that, symbolic execution is applied on this path to obtain path constraints. These path constraints are modified by fixing a set of features. The modified constraints are solved by using an SMT-Solver such as Z3 (De Moura and Bjørner, 2008) to get a solution. This method could generate an adversarial example with a very small  $L_0$ -norm. However, this method consumes a large computational cost due to the symbolic execution and the usage of SMT-Solvers.

**Adversarial Example Improvement.** To the best of our knowledge, most works in this field do not consider the improvement of adversarial examples as an independent phase. Instead, they add the objective of  $L_p$ -norm minimization to the objective function such as box-constrained L-BFGS (Szegedy et al., 2014), Carnili-Wagner (Carlini and Wagner, 2016), ATN (Baluja and Fischer, 2017), etc. Our method could be considered as the second phase of the adversarial example generation methods. The proposed method could be used to enhance the quality of adversarial examples generated by any attack.

## 8 CONCLUSION

We have presented a method to improve the quality of adversarial examples in terms of  $L_0$ -norm and  $L_2$ -norm. The proposed method includes two phases namely *the autoencoder training phase* and *the improvement phase*. In the autoencoder training phase, the proposed method trains an autoencoder to learn how to detect redundantly adversarial features. In the improvement phase, the autoencoder improves the quality of a new adversarial example to generate the first version of the improvement. After that, the first improvement version is fed into the greedy improvement step to enhance more. The experiments are conducted on the MNIST dataset and the CIFAR-10 dataset. The proposed method could enhance the  $L_0$ -norm and  $L_2$ -norm significantly. Additionally, the proposed method could improve the quality of new adversarial examples with a low computational cost. It means that the proposed method could be applied in practice.

In the future, we would investigate more deeply

the impact of high-quality adversarial examples on the performance of the adversarial defence. Besides, we would evaluate the effectiveness of the proposed method with different kinds of autoencoder such as denoising autoencoder, sparse autoencoder, variational autoencoder, symmetric autoencoder, etc. Finally, we would investigate the effectiveness of the proposed method with the adversarial examples generated by other methods such as Carnili-Wagner, DeepFool, DeepCheck, etc.

## ACKNOWLEDGEMENTS

This work has been supported by VNU University of Engineering and Technology under project number CN21.09.

Duc-Anh Nguyen was funded by Vingroup JSC and supported by the Master, PhD Scholarship Programme of Vingroup Innovation Foundation (VINIF), Institute of Big Data, code VINIF.2021.TS.105.

Kha Do Minh was funded by Vingroup JSC and supported by the Master, PhD Scholarship Programme of Vingroup Innovation Foundation (VINIF), Institute of Big Data, code VINIF.2021.ThS.24.

## REFERENCES

- Baluja, S. and Fischer, I. (2017). Adversarial transformation networks: Learning to generate adversarial examples.
- Carlini, N. and Wagner, D. A. (2016). Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644.
- Ciregan, D., Meier, U., and Schmidhuber, J. (2012). Multicolumn deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649.
- De Moura, L. and Bjørner, N. (2008). Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08/ETAPS'08*, pages 337–340, Berlin, Heidelberg. Springer-Verlag.
- Eniser, H. F., Gerasimou, S., and Sen, A. (2019). Deepfault: Fault localization for deep neural networks. *CoRR*, abs/1902.05974.
- Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples.
- Gopinath, D., Păsăreanu, C. S., Wang, K., Zhang, M., and Khurshid, S. (2019). Symbolic execution for attribution and attack synthesis in neural networks. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE '19*, page 282–283. IEEE Press.

- Krizhevsky, A. (2012). Learning multiple layers of features from tiny images. *University of Toronto*.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90.
- Kurakin, A., Goodfellow, I. J., and Bengio, S. (2016). Adversarial examples in the physical world. *CoRR*, abs/1607.02533.
- Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.
- Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998b). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Moosavi-Dezfooli, S., Fawzi, A., and Frossard, P. (2015). Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599.
- Sultana, F., Sufian, A., and Dutta, P. (2019). Advancements in image classification using convolutional neural network. *CoRR*, abs/1905.03288.
- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks.
- Zhang, J., Harman, M., Ma, L., and Liu, Y. (2019). Machine learning testing: Survey, landscapes and horizons.

