Improving Diversity and Quality of Adversarial Examples in Adversarial Transformation Network

Duc-Anh Nguyen
VNU UET FIT: VNU University of Engineering and Technology Faculty of Information Technology
Kha Do Minh
VNU UET FIT: VNU University of Engineering and Technology Faculty of Information Technology
Khoi Nguyen Le
VNU UET FIT: VNU University of Engineering and Technology Faculty of Information Technology
Minh Nguyen Le
Japan Advanced Institute of Science and Technology: Hokuriku Sentan Kagaku Gijutsu Daigakuin Daigaku
Pham Ngoc Hung (hungpn@vnu.edu.vn)
VNU UET FIT: VNU University of Engineering and Technology Faculty of Information Technology
https://orcid.org/0000-0002-5584-5823

Research Article

Keywords: adversarial example, robustness, stacked convolutional autoencoder, targeted attack

Posted Date: January 5th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-868209/v1

Improving Diversity and Quality of Adversarial Examples in Adversarial Transformation Network

Duc-Anh Nguyen[1], Kha Do Minh[1], Khoi Nguyen Le[1], Minh Nguyen Le[2] and Pham Ngoc Hung[1*]

[1*] VNU University of Engineering and Technology (VNU-UET), Buillding E3, 144 Xuan Thuy road, Cau Giay district, Hanoi, Vietnam .
[2]School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), ASAHIDAI 1-1, Nomi 923-1211, Japan.

*Corresponding author(s). E-mail(s): hungpn@vnu.edu.vn;
Contributing authors: nguyenducanh@vnu.edu.vn; 17020827@vnu.edu.vn;
khoi.n.le@vnu.edu.vn; nlminh@jaist.ac.jp;

Abstract

This paper proposes a method to mitigate two major issues of Adversarial Transformation Networks (ATN) including the low diversity and the low quality of adversarial examples. In order to deal with the first issue, this research proposes a stacked convolutional autoencoder based on pattern to generalize ATN. This proposed autoencoder could support different patterns such as *all-feature pattern*, *border feature pattern*, and *class model map pattern*. In order to deal with the second issue, this paper presents an algorithm to improve the quality of adversarial examples in terms of $\boldsymbol{L_0}$-norm and $\boldsymbol{L_2}$-norm. This algorithm employs an adversarial feature ranking heuristics such as JSMA and COI to prioritize adversarial features. To demonstrate the advantages of the proposed method, comprehensive experiments have been conducted on the MNIST dataset and the CIFAR-10 dataset. For the first issue, the proposed autoencoder can generate diverse adversarial examples with the average success rate above 99%. For the second issue, the proposed algorithm could not only improve the quality of adversarial examples significantly but also maintain the average success rate. In terms of $\boldsymbol{L_0}$-norm, the proposed algorithm could decrease from hundreds of adversarial features to one adversarial feature. In terms of $\boldsymbol{L_2}$-norm, the proposed algorithm could reduce the average distance considerably. These results show that the proposed method is capable of generating high-quality and diverse adversarial examples in practice.

Keywords: adversarial example, robustness, stacked convolutional autoencoder, targeted attack

1 Introduction

Convolutional Neural Networks (CNNs) are usually applied to classify images (Sultana et al., 2019). From a labeled dataset consisting of images and their labels, CNNs would be trained to learn important features on this dataset. The trained CNNs are able to predict the label of new images in the same category as the labeled dataset. However, even CNNs achieve high accuracy on the training set and the test set, the resulting models could be failed when applied in practice (Moosavi-Dezfooli et al., 2015; Pei et al., 2017; Su et al., 2017). A reasonable explanation for this issue

is that the training process only focuses on the correctness of the CNNs in terms of accuracy, precision, or F1-score. Meanwhile, inputs from real-world situations could contain perturbation, which can be rarely existed on the training set and the test set. An attacker could create perturbational inputs to interfere the trained models to behave unexpectedly. Therefore, it is important to evaluate the behaviors of CNNs in the presence of perturbation.

Robustness is one of the popular measurements to evaluate the quality of CNNs in the presence of perturbation (Zhang et al., 2019; IEEE, 1990; Carlini and Wagner, 2016; Baluja and Fischer, 2017). To ensure the robustness of CNNs, adversarial example generation is a popular approach. There are two main types of this approach including untargeted attack and targeted attack (Carlini and Wagner, 2016; Akhtar et al., 2021). These two types of attacks aim to modify a correctly predicted input intentionally. For simplicity, this research names this correctly predicted input *attacking image*. The output of an attack is an adversarial example classified as a target label. The target label is any label except for the ground truth label. The main difference between the untargeted attack and the targeted attack is the target label. While the target label is not fixed in the untargeted attack, the target label is a specific label in the targeted attack.

In the targeted attack, autoencoder-based attack is a promising approach to generate adversarial examples. This approach was firstly proposed in Adversarial Transformation Networks (ATN) (Baluja and Fischer, 2017). The general idea is that an autoencoder is trained from an attacked model, a set of attacking images, and a target label. The main advantage of ATN is that the trained autoencoder could be reused for generating adversarial examples from new attacking images with extremely low computational cost. However, ATN has two major disadvantages including *the low diversity of adversarial examples* and *the low quality of adversarial examples*.

Firstly, ATN usually generates adversarial examples with low diversity. In particular, this method only modifies all features to produce adversarial examples from attacking images. However, some specific regions of attacking images could be modified to generate adversarial examples while keeping the remaining regions unchanged. For example, on the MNIST dataset (Lecun et al., 1998b), some regions of the attacking images may contain small noises due to the low quality of cameras. These noises could look like dust in human eyes. In DeepXplore (Pei et al., 2017), they claim that testing domain-specific constraints is important. They could add some black rectangles to any regions of the attacking images to generate adversarial examples.

Secondly, ATN usually generates low-quality adversarial examples. The purpose of ATN is to generate adversarial examples close to the attacking images. Their differences are measured by using L_2-norm. These distances are small if the adversarial examples are close to their corresponding attacking images. A stacked convolutional autoencoder can be utilized to generate adversarial examples from the attacking images. However, some layers on the stacked convolutional autoencoder may be non-linear such as ReLU activation. The adversarial examples may be significantly different from their corresponding attacking images. In the worst case, perturbations are added to all features of attacking images. As a result, the corresponding adversarial examples could be very different from their corresponding attacking images. Therefore, it is worthwhile to improve the quality of adversarial examples generated by ATN.

This paper proposes a method named Pattern-based Adversarial Tranformation Autoencoder (`pbATN`) to address the two issues. Firstly, to deal with the low diversity of adversarial examples, a pattern-based stacked convolutional autoencoder named $\texttt{pbATN}_G$ is proposed in `pbATN`. $\texttt{pbATN}_G$ is a generalization of ATN. Similarly to ATN, the generated adversarial examples should be close to the attacking images in terms of L_2-norm. The proposed $\texttt{pbATN}_G$ aims to modify a set of specific features, which is defined by a pattern. This research suggests several patterns including *all-feature pattern*, *border feature pattern*, and *class model map pattern*. Among these patterns, *all-feature pattern* is used in ATN. Secondly, to improve the quality of adversarial examples, an improvement algorithm named $\texttt{pbATN}_I$ is proposed in `pbATN`. $\texttt{pbATN}_I$ is able to improve the quality of adversarial examples in terms of both L_0-norm and L_2-norm. The main idea of $\texttt{pbATN}_I$ is to restore *redundantly adversarial features* in an adversarial example to the original values in the corresponding attacking image. An adversarial feature is redundant

if it does not contribute to the decision of the attacked model, which is a predicted label. This research proposes to use JSMA (Papernot et al., 2015) and COI (Gopinath et al., 2019) to rank features in terms of their impact on the decision of the attacked model. As a result, L_0-norm and L_2-norm between the improved adversarial example and its attacking image could be reduced significantly.

The rest of this paper is organized as follows. Section 2 delivers the overview of related research on adversarial example generation. Section 3 provides the background of CNNs and the targeted attack. The overview of `pbATN` is shown in Section 4. Next, Section 5 presents the experiment to demonstrate the advantages of `pbATN`. The discussion is presented in Section 6. Finally, the conclusion is described in Section 7.

2 Related Works

This section presents an overview of related researches as follows. Firstly, some well-known adversarial example generation methods for CNNs are presented. Secondly, outstanding researches on finding saliency map of CNNs are discussed.

Adversarial example generation: Many adversarial example generation methods are proposed to evaluate the robustness of CNNs. These methods would modify an attacking image to generate an adversarial example. In the case of the targeted attack, the label of this adversarial example is a specific label differing from the ground truth label of the attacking image. In the case of the untargeted attack, the label of this adversarial example is any label except the ground truth label of the attacking image.

In order to evaluate the quality of an adversarial example, L_0-norm, L_2-norm, and L_∞-norm are commonly used. Several popular methods using these metrics could be referred to box-constrained L-BFGS (Szegedy et al., 2014), DeepFool (Moosavi-Dezfooli et al., 2015), targeted FGSM (Goodfellow et al., 2015), Carnili-Wagner (Carlini and Wagner, 2016), BIS (Kurakin et al., 2016), ATN (Baluja and Fischer, 2017), MI-FGSM (Dong et al., 2017), and DeepCheck (Gopinath et al., 2019). Apart from DeepCheck, the other methods could modify any features of attacking images to generate adversarial examples. The attacked CNNs would predict these adversarial examples incorrectly. In the worst case, perturbations are added to all features of attacking images. Concerning DeepCheck, this method focuses on modifying the top-n most important features of attacking images to generate adversarial examples. DeepCheck is capable of modifying top-1 feature on the attacking image to produce adversarial examples successfully. Inspired by the mentioned methods, our proposed `pbATN` uses L_0-norm and L_2-norm metric to evaluate the quality of adversarial examples. To generate diverse and high-quality adversarial examples, `pbATN` includes two phases named $\texttt{pbATN}_G$ and $\texttt{pbATN}_I$. $\texttt{pbATN}_G$ is used to generate diverse adversarial examples. $\texttt{pbATN}_G$ generalizes ATN to modify a set of specific features on the attacking image. The experiments show that $\texttt{pbATN}_G$ could generate adversarial examples with high average success rate. $\texttt{pbATN}_I$ is employed to improve quality of adversarial examples. The experiments show that $\texttt{pbATN}_I$ could help to modify one feature on attacking images to generate adversarial examples. This result is competitive to DeepCheck. Additionally, $\texttt{pbATN}_I$ could be used to improve the adversarial quality of adversarial examples generated by targeted FGSM, box-constrained L-BFGS, and Carnili-Wagner.

Besides these aforementioned distance metrics, some other works propose additional metrics to generate adversarial examples. These metrics use the internal states of CNNs (i.e., state of neurons, state of layers, etc.). This research calls them *state-based metric*. These state-based metrics could be combined with L_p-norm. Several state-based metrics could be referred to adversarial distance (Papernot et al., 2015), average robustness (Moosavi-Dezfooli et al., 2015), adversarial severity (Bastani et al., 2016), adversarial frequency (Bastani et al., 2016), point-wise robustness (Bastani et al., 2016), local/global adversarial robustness (Katz et al., 2017), neuron coverage (Pei et al., 2017), a set of multi-granularity testing criteria (Ma et al., 2018), Lipschitz continuity (Sun et al., 2018b), sign-sign coverage (Sun et al., 2018a), and probabilistic robustness (Mangal et al., 2019).

Some other works do not use L_p-norm or *state-based metric* to generate adversarial examples. Instead, these works focus on generating natural adversarial examples. They argue that in reality, many natural adversarial examples could be

very different from their corresponding attacking images. Using L_p-norm or *state-based metric* usually do not satisfy this argument. Brown et al. (2018) firstly propose an approach to generate unrestricted adversarial examples. Their approach does not consider L_p-norm to generate adversarial examples. Alcorn et al. (2018) apply natural transformation such as changing viewpoint, lighting, coloring. Naderi et al. (2021) propose a method based on geometric transformations to generate natural perturbations. Their transformations include scaling, rotation, shear, and translation which are performed on the attacking image.

Saliency map generation: Saliency map generation draws a great interest from research groups. Saliency map describes the impact of features on the classification of a CNN (Simonyan et al., 2013; Gu and Tresp, 2019). The saliency map usually has the same size as the input of the CNN.

Simonyan et al. (2013) clarify two main types of saliency map known as image-specific class saliency map and class model map. An image-specific class saliency map represents the influence of features of a specific image on the classification of the CNN. Differing from image-specific class saliency map, a class model map could be used to explain how the CNN classifies a set of images as a specific label. Similarly to the idea of Simonyan et al. (2013), Zeiler and Fergus (2013) use a multi-layered deconvolutional network to produce a saliency map. Springenberg et al. (2014) propose guided backpropagation, which is better than gradient-based technique proposed in Simonyan et al. (2013). Papernot et al. (2015) propose Jacobian-based saliency map attack to construct image-specific class saliency map of an input image. Cao et al. (2015) present a novel feed-back convolutional neural network architecture to capture the high-level semantic concepts of an image, then project the obtained information into salience maps. Zhang et al. (2016) introduce excitation backprop to enhance the limitation of gradient-based technique. Fong and Vedaldi (2017) attempt to find the portion of an image, which has the largest impact on the decision of the CNN. Unlike other saliency map generation methods, their method explicitly modifies the attacking image to generate an adversarial example. The modification of this attacking image could be interpretable to human observers. Dabkowski and Gal (2017) develop a fast saliency map detection method to train a model. This model could predict a saliency map for any attacking image with a low computational cost. Yu et al. (2018) propose a method to predict saliency map of an input image with low computation cost.

The relationship between saliency map and adversarial example is discussed in many works. Fong and Vedaldi (2017) and Gu and Tresp (2019) claim that saliency maps can be used to explain adversarial example classifications. Tsipras et al. (2019) find out that more robust models have more interpretable saliency maps. Etmann et al. (2019) quantify the relationship between saliency map and attacking image by analyzing their alignment. They conclude that the more linear CNN has the stronger connection between its robustness and alignment.

Due to the existence of the relationship between saliency map and adversarial example, our research utilizes the class model map (Simonyan et al., 2013) as a pattern in $\texttt{pbATN}_G$. By generating a class model map of an attacking image set, $\texttt{pbATN}_G$ could find top-n important features which have the largest impact on the classification of the CNN. $\texttt{pbATN}_G$ then modifies these important features to generate adversarial examples.

3 Background

3.1 Convolutional Neural Network

Definition 1 (Convolutional Neural Network (CNN)) A typical CNN $\boldsymbol{M}$ is a tuple $(\boldsymbol{L}, \boldsymbol{W}, \boldsymbol{\theta})$, where $\boldsymbol{L} = \{L_k \mid k \in \{0, ..., h-1\}\}$ is a set of layers (i.e., a convolutional layer, a pooling layer, an activation layer, sampling layer, a fully-connected layer, etc.), h is the number of layers, $\boldsymbol{W} \subseteq \boldsymbol{L} \times \boldsymbol{L}$ is a set of layers, and $\boldsymbol{\theta} = \{\theta_0, \theta_1, ..., \theta_{h-1}\}$ represents a set of activation functions.

The activation function of the k-th layer, denoted by θ_k, is either linear or non-linear. The most popular activation functions are softmax, sigmoid, hyperbolic tangent, ReLU, and leaky ReLU (Goodfellow et al., 2016). The activation function of the output layer is softmax. A typical CNN $\boldsymbol{M}$ can be illustrated in a simple representation as $\boldsymbol{M}(\mathbf{x}) = L_{h-1} \circ L_{h-2} \circ \ ... \ \circ L_0$, where $\circ$ presents linear connection, $\mathbf{x}$ is the input

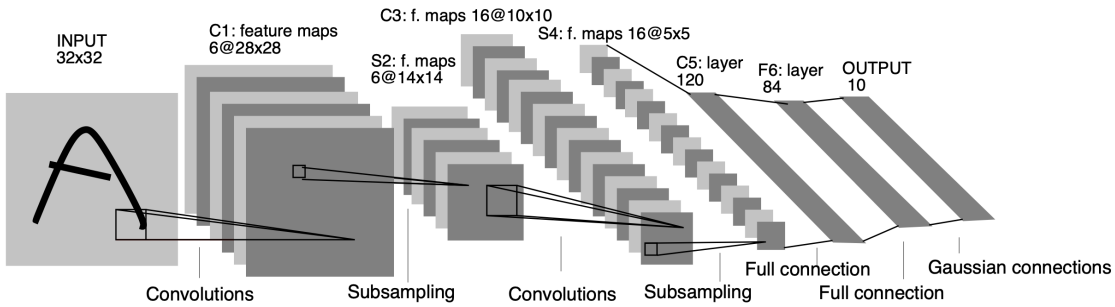


Fig. 1: Architecture of LeNet-5 (Lecun et al., 1998a).

image, L_0 is the input layer, and L_{h-1} is the output layer. The input image has the shape $(width, height, channel)$. The number of features on L_0 is computed by $width \times height \times channel$. If the input image is 2D, the size of channel is one. If the input image is 3D, the size of channel is three corresponding to red, green, and blue channel. Other CNNs could have more complicated representation such as ResNet (He et al., 2015) and Inception (Szegedy et al., 2015).

For example, Fig. 1 illustrates the architecture of LeNet-5 (Lecun et al., 1998a), which is a CNN model. This model is designed for handwritten and machine-printed character recognition. This model comprises 8 layers. Convolutional layers are labeled Ci, where i is layer index. Sub-sampling layers and fully-connected layers are labeled Si and Fi, respectively. The shape of the input image is (32, 32, 1). Layer C1 is a convolutional layer with 6 feature maps of size 28×28. Layer S2 is a sub-sampling layer with 6 feature maps of size 14×14. Layer C3 is a convolutional layer with 16 feature maps of size 10×10. Layer S4 is a sub-sampling layer with 16 feature maps of size 5×5. Layer C5 is a convolutional layer with 120 feature maps. The size of a feature map in this layer is 1×1. Layer C5 is then flatten into fully-connected layer F6 with 84 hidden units. The output layer has 10 outputs in which each output is corresponding to a digit.

Let $\mathbf{X}$ be the training set containing a set of images. Let k be the number of labels. The predicted probability vector of an attacking image $\mathbf{x}^{w \times h \times s} \in \mathbf{X}$ is denoted by $\boldsymbol{M}(\mathbf{x}) \in \mathbb{R}^k$, where w is corresponding to width, h is corresponding to height, and c is corresponding to channel. The probability of label c is denoted by $\boldsymbol{M}_c(\mathbf{x}) \in \mathbb{R}$. Let $\frac{\partial \boldsymbol{M}_c(\mathbf{x})}{\partial x_i}$ be the gradient of the c-th output neuron with respect to the feature x_i. The ground truth label of $\mathbf{x}$ is denoted by y_{true}.

3.2 Targeted Attack

If an input image is used in the targeted attack, this research uses the name *attacking image* alternatively. The attacking image must be predicted correctly by the attacked CNN. Let $\mathbf{x}'$ be an adversarial example generated from the attack. The predicted probability vector of $\mathbf{x}'$ is denoted by $\boldsymbol{M}(\mathbf{x}')$.

The key idea of a targeted attack method is that perturbations are added to an attacking image to generate an adversarial example. This adversarial example is classified as a target label $y^* \neq y_{true}$. The neuron corresponding to the target label on the output layer is called *target neuron*. In this research, a modified feature is called *adversarial feature*. This research defines two types of perturbations, namely *mandatory perturbation* and *redundant perturbation*:

Definition 2 (Adversarial feature) Given an attacking image $\mathbf{x}$ and its corresponding adversarial example $\mathbf{x}'$, a feature x_i' on the adversarial example $\mathbf{x}'$ is adversarial if and only if $x_i' \neq x_i$.

Definition 3 (Mandatory perturbation) A perturbation β is mandatory if and only if $\mathbf{x}$ is predicted correctly by the attacked model and $(\mathbf{x} + \beta)$ produces an adversarial example classified as the target label y^*.

Definition 4 (Redundant perturbation) A perturbation β is redundant if and only if $\mathbf{x}$ and $(\mathbf{x} + \beta)$ are classified as the ground true label.

A feature containing *redundant perturbation* is called *redundantly adversarial feature*. Redundant perturbations play an important role in our proposed $\texttt{pbATN}_I$. We would find out *redundant perturbation* and remove them from the adversarial examples. This leads to an improvement in the quality of adversarial examples. The detail of $\texttt{pbATN}_I$ is presented in Section 4.2.

3.2.1 Success Rate

In addition to L_p-norm, success rate is a common metric to evaluate the effectiveness of an adversarial example generation method. Let $sr(\boldsymbol{M}) \in [0, 1]$ be a success rate of an adversarial example generation method used to attack a CNN model $\boldsymbol{M}$. Ideally, an adversarial example generation method should produce a set of adversarial examples with small L_p-norm and achieve high success rate. The success rate of an adversarial example generation method is computed as follows:

$$sr(\boldsymbol{M}) = \frac{\sum_{\mathbf{x}'} \mathbb{1}(\text{argmax}(\boldsymbol{M}(\mathbf{x}')) = y^*)}{\mid \mathbf{X}_{attack} \mid} \quad (1)$$

where $\mathbf{X}_{attack}$ is a set of attacking images, $\mathbf{x}'$ is an adversarial example generated by modifying an attacking sample on $\mathbf{X}_{attack}$, and $\mathbb{1}$ is a indicator function. Function $\mathbb{1}(\text{argmax}(\boldsymbol{M}(\mathbf{x}')) = y^*)$ returns 1 if the model $\boldsymbol{M}$ classifies $\mathbf{x}'$ as the target label y^*, and returns 0 otherwise.

3.2.2 L_p-norm

Given an attacking image $\mathbf{x}$, its adversarial example $\mathbf{x}'$ should be similar to the corresponding attacking image. L_p-norm is one popular way to measure the difference between an attacking image and its corresponding adversarial example. The equation of L_p-norm is defined as follows:

$$L_p(\mathbf{x}, \mathbf{x}') = (\sum_i (x_i - x_i')^p)^{\frac{1}{p}} \quad (2)$$

The popular metrics of L_p-norm are L_0-norm (p=0), L_2-norm (p=2), and L_∞-norm (p=∞). L_0-norm is known as Hamming distance, which is interpreted as the number of different features when comparing $\mathbf{x}$ and $\mathbf{x}'$. $L_0(\mathbf{x}, \mathbf{x}') \approx 0$ if and only if both $\mathbf{x}$ and $\mathbf{x}'$ are very similar. L_2-norm is known as Euclidean distance. It is calculated as the Euclidean distance from $\mathbf{x}$ to $\mathbf{x}'$. L_∞-norm measures the maximum change in value of features in every axis of the coordinate.

There are a wide range of attack methods using L_p-norm. Targeted FGSM (Goodfellow et al., 2015) and BIM (Kurakin et al., 2016) generate adversarial examples to minimize L_∞-norm. Box-constrained L-BFGS (Szegedy et al., 2014), Carlini-Wagner L_2 (Carlini and Wagner, 2016), and ATN (Baluja and Fischer, 2017) aim to minimize L_2-norm. DeepFool (Moosavi-Dezfooli et al., 2015), Carnili-Wagner L_0 (Carlini and Wagner, 2016), and DeepCheck (Gopinath et al., 2019) minimize L_0-norm.

3.3 Autoencoder

Generally, an autoencoder consists of one encoder and one decoder (Bengio et al., 2006; Masci et al., 2011). Several popular variations of autoencoder are denoising (Vincent et al., 2010), sparse (Ng, 2011), convolutional (Masci et al., 2011), variational (Kingma and Welling, 2014), symmetric (Pu et al., 2017), Wasserstein (Tolstikhin et al., 2019), etc. They differ on the architecture of the encoder and decoder, on the computation of the loss, or on the noise inserted into the network. In the context of image reconstruction, the input and the output of the autoencoder are called *input image* and *reconstructed image*, respectively. This research uses stacked convolutional autoencoder to generate adversarial examples. The conventional autoencoders including sparse autoencoder and stacked autoencoder are presented in detail.

3.3.1 Sparse Autoencoder

A sparse autoencoder has one input layer, one hidden layer, and one output layer. This autoencoder is the simplest autoencoder. This autoencoder takes an input image $\mathbf{x}_{in} \in \mathbb{R}^{m \times 1}$. In the encoder stage, this autoencoder maps this input image to a latent representation $\mathbf{z} \in \mathbb{R}^{n \times 1}$ in which $n < m$:

$$\mathbf{z} = f(\mathbf{W}_1^T \cdot \mathbf{x}_{in} + \mathbf{b}_1) \quad (3)$$

where $\mathbf{b}_1 \in \mathbb{R}^{n \times 1}$ is the bias matrix of the hidden layer, $\mathbf{W}_1 \in \mathbb{R}^{m \times n}$ is the weight matrix between the input layer and the hidden layer, and $f(.)$ is an activation function.

In the decoder stage, the latent representation $\mathbf{z}$ is then transformed into the input image

as follows:

$$\mathbf{x}_{out} = g(\mathbf{W}_2^T \cdot \mathbf{z} + \mathbf{b}_2) \quad (4)$$

where $\mathbf{x}_{out}$ is the reconstructed image, $g(.)$ is an activation function, $\mathbf{b}_2 \in \mathbb{R}^{m \times 1}$ is the bias matrix of the output layer, and $\mathbf{W}_2 \in \mathbb{R}^{n \times m}$ is the weight matrix between the hidden layer and the output layer.

This reconstructed image should be similar to the input image. To fulfil this requirement, the popular objective function of this autoencoder is as follows:

$$\sum_{\mathbf{x}_{in}} L_2^2(\mathbf{x}_{in}, \mathbf{x}_{out}) \quad (5)$$

3.3.2 Stacked Autoencoder

Stacked autoencoder is an extension of sparse autoencoder. A stacked autoencoder has at least one hidden layer. In the encoder stage, the size of the previous layer is larger than the size of the next layer. In the decoder stage, the size of the previous layer is smaller than the size of the next layer.

Figure 2 indicates an example of stacked autoencoder. The input layer has 5 neurons. In the encoder stage, there are 2 hidden layers with the size of 4 neurons and 3 neurons. The latent representation has 2 neurons. In the decoder stage, there are 2 hidden layers with 3 neurons and 4 neurons. The output layer has the same size as the input layer. As can be seen, the reconstructed image is similar to the input image.

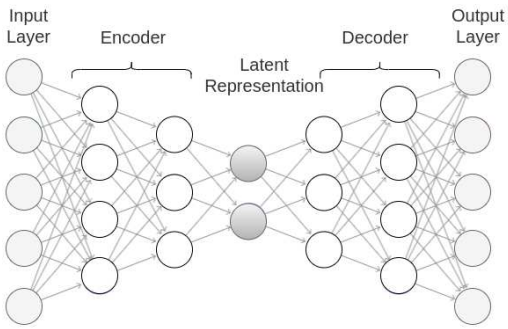


Fig. 2: Example of a stacked autoencoder.

3.3.3 Stacked Convolutional Autoencoder

Stacked autoencoder ignores the 2D/3D image structure because the input image is flatten into a k-dimensional vector. Each dimension represents a pixel of the input image. For example, on the MNIST dataset, a 28 × 28 image is flattened into a 784-dimensional vector. Because the stacked autoencoder does not consider the 2D/3D image structure, this type of autoencoder does not reconstruct the input image effectively (Masci et al., 2011). To mitigate this issue, the stacked convolutional autoencoder is proposed. In the encoder stage, a layer can be convolutional, down-sampling, and fully-connected. In the decoder stage, a layer can be deconvolutional, up-sampling, and fully-connected. The typical objective function of the stacked convolutional autoencoder is identical to Equation 5.

Figure 3 illustrates an example of stacked convolutional autoencoder. The input image is a 2D image with the size 28 × 28. In the encoder stage, this image is fed into a convolutional layer with the stride of value 2. The resulting layer $Conv1$ consists of 32 feature maps of size 14 × 14. At the end of the encoder, the layer is flattened into a fully-connected layer of size 10 × 1. The fully-connected layer is the latent representation. In the decoder stage, this latent representation is fed into a fully-connected layer (i.e., FC), an upsampling layer (i.e., $Reshape$) and deconvolutional layers (i.e., $DeConv3$, $DeConv2$, and $DeConv1$) to reconstruct the input image.

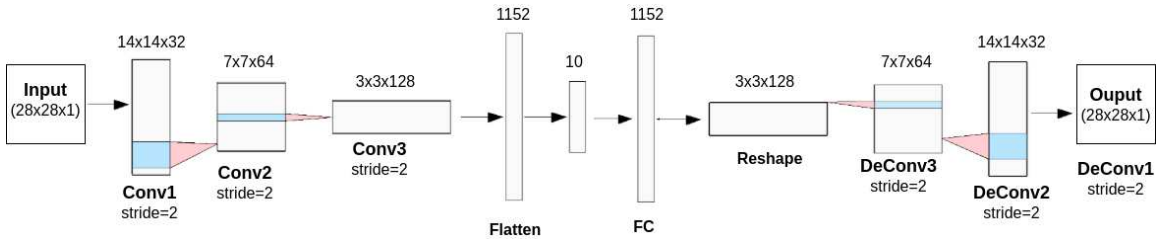


Fig. 3: An example of stacked convolutional autoencoder.

3.4 Popular Adversarial Example Generation

3.4.1 Box-constrained L-BFGS

Szegedy et al. (2014) propose box-constrained L-BFGS to generate adversarial examples by solving the following box-constrained optimization problem:

$$\begin{aligned} &\text{minimize } (1-c)\cdot L_2^2(\mathbf{x}', \mathbf{x}) + c \cdot f(y^*, \boldsymbol{M}(\mathbf{x}')) \\ &\text{such that } \mathbf{x}' \in [0,1]^{w \times h \times s} \end{aligned} \tag{6}$$

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

3.4.2 Targeted FGSM

Goodfellow et al. (2015) propose targeted FGSM to generate adversarial examples which are similar to the attacking images in terms of L_∞-norm. Targeted FGSM generates adversarial examples by modifying all features:

$$\mathbf{x}' = \mathbf{x} - \omega \cdot sign\left(\frac{\partial \boldsymbol{M}_{y^*}(\mathbf{x})}{\partial x_i}\right) \tag{7}$$

where function $sign(.)$ returns the sign of $\frac{\partial \boldsymbol{M}_{y^*}(\mathbf{x})}{\partial x_i}$ and ω is a positive value to shift values of all features in $\mathbf{x}$.

Main disadvantage of targeted FGSM is that its effectiveness is sensitive to the value of ω. If the value of ω is large, the success rate of the targeted FGSM could be high. However, adversarial examples could be very different from the corresponding attacking images in terms of L_p-norm. Otherwise, if the value ω is very small, the success rate of the targeted FGSM could be low significantly. In terms of L_p-norm, adversarial examples could be very similar from the corresponding attacking images.

3.4.3 Carnili-Wagner L_2

Inspired from box-constrained L-BFGS, Carlini and Wagner (2016) propose a method to minimize L_2-norm. An adversarial example is generated by solving the following equation:

$$\begin{aligned} &\text{minimize } (1-c)\cdot L_2^2(\mathbf{x}', \mathbf{x}) + c \cdot f(y^*, \boldsymbol{M}(\mathbf{x}')) \\ &\text{such that } \mathbf{x}' \in [0,1]^{w \times h \times s} \end{aligned} \tag{8}$$

where f is their suggested objective function:

$$f(\mathbf{x}') = max(max\{Z(\mathbf{x}')_i : \forall i \neq y^*\} - Z(\mathbf{x}')_{y^*}, -k) \tag{9}$$

where $Z(.)$ returns the pre-softmax value of the output layer, term $max\{Z(\mathbf{x}')_i : \forall i \neq y^*\}$ is the largest pre-softmax value except the target neuron, and term $Z(\mathbf{x}')_{y^*}$ is the pre-softmax value of target neuron.

3.4.4 Adversarial Transformation Networks

Baluja and Fischer (2017) introduce ATN to generate adversarial examples based on stacked convolutional autoencoder. The input and output of ATN are attacking image and adversarial example, respectively. The authors suggest using L_2-norm to compute the distance between the adversarial example and its corresponding attacking image. The loss function of ATN is as follows:

$$\sum_{\mathbf{x}} \beta \cdot L_2(\mathbf{x}, \mathbf{x}') + L_2(\boldsymbol{M}(\mathbf{x}'), r_\alpha(\boldsymbol{M}(\mathbf{x}), y^*)) \tag{10}$$

where β is a weight to balance two terms and function $r_\alpha(.)$ modifies $\boldsymbol{M}(\mathbf{x})$ with the expectation that the modification on the attacking image $\mathbf{x}$ is the least:

$$\begin{aligned} &r_\alpha(\boldsymbol{M}(\mathbf{x}), y^*) \\ &= n\left(\left\{ \begin{array}{ll} \alpha \cdot max(\boldsymbol{M}(\mathbf{x})) & if \quad i = y^* \\ \boldsymbol{M}(\mathbf{x})_i & otherwise \end{array} \right\}_{i \in \{0..k-1\}}\right) \end{aligned} \tag{11}$$

where α is greater than 1 and $n(.)$ normalizes a vector into an array of probabilities.

However, this method does not support modifying specific portions of an attacking image to generate an adversarial example. Instead, this method adds perturbation to all features of the attacking image. Adding perturbation to all features could be consider as a pattern. This paper generalizes ATN to support different patterns, in which a pattern defines a set of modified features.

4 Pattern-based Adversarial Transformation Network

To generate high-quality and diverse adversarial examples, pbATN consists of two phases named $\texttt{pbATN}_G$ and $\texttt{pbATN}_I$. The illustration of pbATN is shown in Fig. 4. In the first phase, $\texttt{pbATN}_G$ is applied to produce a set of adversarial examples based on patterns. However, these adversarial examples could not be high-quality since they could contain *redundantly adversarial features*. By removing these features, the resulting adversarial examples are still classified as a target label. Additionally, they are more similar to the attacking images than the prior adversarial examples.

It is suggested that these adversarial examples should be close to their corresponding attacking images. Therefore, in the second phase, $\texttt{pbATN}_I$ is employed to improve the quality of these adversarial examples. The proposed $\texttt{pbATN}_I$ finds out *redundantly adversarial features* and restores them to the original values.

4.1 Pattern-based Stacked Convolutional Autoencoder

In the first phase of pbATN, $\texttt{pbATN}_G$ is applied to produce a set of adversarial examples based on stacked convolutional autoencoders. These adversarial examples must satisfy a specific pattern. The objective of $\texttt{pbATN}_G$ satisfies two requirements. The first requirement is that the adversarial example should be similar to the attacking image as much as possible. The second requirement is that the labels of adversarial examples are the target label y^*. Based on these two requirements, the objective of $\texttt{pbATN}_G$ is defined as follows:

$$\sum_{\mathbf{x}} (1-\epsilon) \cdot L_2^2(\gamma(\mathcal{P}, \mathbf{x}), \mathbf{res}) + \epsilon \cdot \text{CE}(y^*, \boldsymbol{M}(\mathbf{x}')) \tag{12}$$

where $\mathcal{P}$ is the pattern, $\boldsymbol{M}$ is the attacked model, CE is cross-entropy, and **res** is the reconstructed image and computed by $ae_{y^*}(\gamma(\mathcal{P}, \mathbf{x}))$ (ae_{y^*} is the training autoencoder).

Compared to Equation 10, Equation 12 has the following differences:

- *The weight between terms*: In our objective function, the used weights are $(1-\epsilon)$ for the first term and ϵ for the second term. In ATN, the first term has the weight β and the second term has the weight 1. Essentially, our configured weights are similar to the weights of ATN. We set up this type of weight to have a better intuition about the importance of terms.
- *Crossentropy*: In the second term of Equation 10, while ATN uses L_2-norm, pbATN uses crossentropy function. This would lead to a better optimization of Equation 12.
- *Pattern*: In Equation 10, all features could be modified. Otherwise, in Equation 12, only features satisfying a pattern $\mathcal{P}$ would be modified.
- *Function* r_α: This function is used in Equation 10. Equation 12 does not use function r_α. We only consider this function as an optional choice. We find out that using r_α does not produce a better result than not using r_α when attacking CNN models based on patterns. Instead of using function r_α, this paper uses a general strategy by proposing $\texttt{pbATN}_I$, which improves the quality of adversarial examples in terms of L_0-norm and L_2-norm.

The function $\gamma(\mathcal{P}, \mathbf{x})$ creates a pattern map of $\mathbf{x}$ satisfying $\mathcal{P}$. Specifically, if the feature $x_i \in \mathbf{x}$ does not satisfied the pattern $\mathcal{P}$, the value of the corresponding element in the pattern map is zero. Otherwise, if the feature x_i satisfies the pattern $\mathcal{P}$, the value of the corresponding element in the pattern map is x_i:

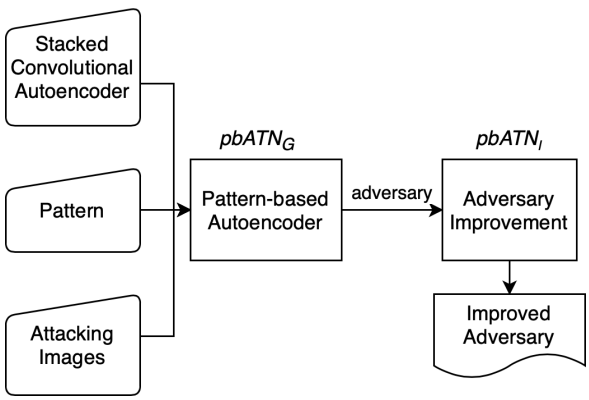


Fig. 4: The overview of pbATN

$$\gamma(\mathcal{P}, x_i) = \begin{cases} x_i & \text{if } x_i \text{ satisfies the pattern } \mathcal{P} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

The adversarial example $\mathbf{x}'$ is computed by using the equation $\mathbf{x} - \mathcal{P}(\mathbf{x}) + \mathcal{P}(\mathbf{res})$. This equation replaces the features on $\mathbf{x}$ satisfying the pattern $\mathcal{P}$ with new values returning from the autoencoder ae_{y^*}. All features not satisfied the pattern $\mathcal{P}$ remain unchanged. This research uses three patterns including *all-feature pattern*, *border feature pattern*, and *class model map pattern*:

All-feature pattern: all features could be modified to generate adversarial examples. ATN uses this pattern to generate adversarial examples.

Border feature pattern: all features locating at the edge of objects on attacking images would be modified. For example, on the MNIST dataset (Lecun et al., 1998b), objects are digits.

Class model map pattern: Let $\mathcal{I}$ be a class model map of attacking images. These attacking images are classified as label c. The shape of $\mathcal{I}$ is the same as the shape of the attacking images. The class model map $\mathcal{I}$ is generated by minimizing the following objective function:

$$\underset{\mathcal{I}}{\operatorname{argmax}}\, S_c(\mathcal{I}) - \lambda \cdot L_2^2(\mathcal{I}) \tag{14}$$

where S_c is the score of the label c, λ is a hyper-parameter, and $\mathcal{I}$ is initialized as a zero image. The class model map $\mathcal{I}$ is constructed by minimizing Equation 14. The authors use back-propagation to find out $\mathcal{I}$.

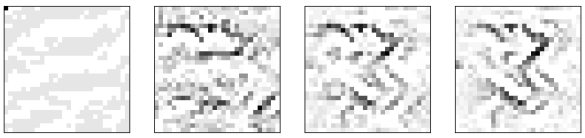


Fig. 5: Example of a class model map during the training process.

An example of training a class model map is illustrated in Fig. 5. This class model map is trained on 1,000 attacking images classified as label 3. These attacking images belong to the MNIST dataset. From leftmost side to rightmost slide, the class model map is captured after 10, 20, 30, and 40 epochs. The value of λ is chosen 0.2. The more dark pixel on $\mathcal{I}$ means that this pixel has more impact on the output neuron corresponding to the label c.

4.2 Adversarial Example Improvement

The adversarial examples generated by $\texttt{pbATN}_G$ might contain *redundant perturbation*. Hence, this section presents an algorithm named $\texttt{pbATN}_I$ to improve the quality of adversarial examples in terms of L_0-norm and L_2-norm. We empirically observed that a large number of the generated adversarial examples from an autoencoder could contain many *redundant perturbations*. If these *redundantly adversarial features* are restored to the original values, the resulting adversarial examples are still classified as the target label y^*. In other words, these *redundantly adversarial features* do not contribute to or only have a very little impact on the final classification of adversarial examples. Additionally, due to the existence of *redundant perturbation*, the quality of adversarial examples would be degraded in terms of L_0-norm and L_2-norm. Meanwhile, one of the most important objectives of the targeted attack is to generate adversarial examples similar to their attacking images as much as possible. Therefore, in order to improve the quality of adversarial examples, the features containing *redundant perturbation* should be restored to the original values.

This research proposes Algorithm 1 to improve the quality of adversarial example in terms of L_0-norm and L_2-norm. The inputs of this algorithm are an adversarial example $\mathbf{x}'$, the attacking image $\mathbf{x}$ of $\mathbf{x}'$, a target label y^*, an attacked model $\boldsymbol{M}$, a step size $\alpha >= 1$, a positive threshold $\delta < \alpha$, and a decay rate $t >= 1$. The output of the algorithm is an improved adversarial example. If α is greater than δ and the restored rate is not converged, the algorithm detects adversarial features by comparing $\mathbf{x}$ and $\mathbf{x}'$, and then stores these features in a set S (line 2). After that, the adversarial features of S are ranked by applying an adversarial feature ranking heuristics (line 3) and stored in S'. The beginning elements of set S' tend to have a low impact on the classification of adversarial example $\mathbf{x}'$. The last elements of S' tend to affect significantly the decision of the attacked model. The proposed algorithm then

Algorithm 1 $\texttt{pbATN}_I$: Improve the quality of adversarial examples in terms of L_0-norm and L_2-norm

Input: adversarial example $\mathbf{x}'$, attacking image $\mathbf{x}$, target label y^*, CNN $\boldsymbol{M}$, step α, threshold δ, and decay rate t

Output: An improved adversarial example

1: **while** $\alpha > \delta$ **do**
2: $S \leftarrow$ COMPUTE_DIFF_FEATURES$(\mathbf{x}, \mathbf{x}')$
3: $S' \leftarrow$ RANK_FEATURES(S)
4: $\#block \leftarrow \lceil |S'|/\alpha \rceil$
5: **for** $idx \leftarrow 0$, $idx < \#block$, $idx \leftarrow idx + 1$ **do**
6: $start \leftarrow \alpha \cdot idx$; $end \leftarrow start + \alpha$
7: **if** $end > |S'|$ **then**
8: $end \leftarrow |S'|$
9: **end if**
10: $\mathbf{x}'_{clone} \leftarrow \mathbf{x}'$
11: $\mathbf{x}'_{start...end-1} \leftarrow \mathbf{x}_{start...end-1}$
12: **if** $\text{argmax}\, \boldsymbol{M}(\mathbf{x}') \neq y^*$ **then**
13: $\mathbf{x}' \leftarrow \mathbf{x}'_{clone}$
14: **end if**
15: **if** $is\_converged$ **then**
16: return $\mathbf{x}'$
17: **end if**
18: **end for**
19: $\alpha \leftarrow \lfloor \alpha/t \rfloor$
20: **end while**
21: **return** $\mathbf{x}'$

restores sequentially adversarial features in S'. A set of these sequentially adversarial features is called *block*. For each block starting from position *start* to position *end*, in which the size of block is step α (line 6 - 10), the algorithm updates the adversarial example $\mathbf{x}'$ by restoring the block (line 11). If the restoration changes the classification, the current adversarial features are reverted to their original values (line 12-14). Next, α is decreased by t times (line 19).

Algorithm 1 terminates when α is not greater than threshold δ (line 1) or the restored rate (denoted by r) converges (line 15). At the i-th modification, the restored rate r of an adversarial example improvement is defined as follows:

$$r(\mathbf{x}, \mathbf{x}', i) = \frac{L_0(\mathbf{x}, \mathbf{x}'_i)}{L_0(\mathbf{x}, \mathbf{x}')} \tag{15}$$

where $\mathbf{x}'_i$ is the improved adversarial example at the i-th modification. At the p-th modification, the restored rate r converges if and only if it satisfies the following equation:

$$\bigwedge (r(\mathbf{x}, \mathbf{x}', i) - r(\mathbf{x}, \mathbf{x}', i-1) <= \beta) \;\mid\; i \in [p-k+1, p] \tag{16}$$

where $k >= 1$ is number of prediction with the improvement of adversarial examples less than β.

The value of step α plays an important role in the performance of our proposed $\texttt{pbATN}_I$. The step α must be greater than or equal to 1, which controls the performance of $\texttt{pbATN}_I$. If step α is equal to one, $\texttt{pbATN}_I$ calls the prediction $\boldsymbol{M}(\mathbf{x}')$ at least $\#block$ times. The cost of making a prediction call is expensive. When $\#block$ is large enough such as in the case of *all-feature pattern*, the total cost of prediction could be extremely huge. Therefore, step α should be large enough to decrease this computational cost.

Adversarial Feature Ranking Heuristics: The purpose of an adversarial feature ranking heuristics (line 3) is to estimate the impact of an adversarial feature on the target neuron. The impact of adversarial feature x'_i on the output neuron y^*-th is denoted by $s_{y^*}(x'_i)$. Adversarial feature x'_i has a higher impact on the output neuron y^* than adversarial feature x'_j if and only if $s_{y^*}(x'_i) > s_{y^*}(x'_j)$.

For the feature x_i on an attacking image, there are several well-known feature ranking heuristics such as JSMA (Papernot et al., 2015), COI (Gopinath et al., 2019), etc. Concerning JSMA, Papernot et al. (2015) propose two heuristics. This research calls them JSMA^+ and JSMA^-. These two ranking heuristics are applied to compute the score of features on the attacking image. Specifically, for the feature x_i which should increase to generate an adversarial example, JSMA^+ assigns this feature to a non-zero score. By contrast, for the feature x_i which should decrease to generate an adversarial example, JSMA^- assigns this feature to a non-zero score. Concerning COI, the score of a feature x_i is computed by the multiplication of its intensity and its gradient.

However, the above heuristics do not consider ranking adversarial features. In this research, we apply these heuristics in a different context. We would compute the scores of adversarial features rather than the scores of features on the attacking images. The score of an adversarial feature x'_i are defined as follows:

- JSMA^{+}:

$$s_{y^*}(x'_i) = \begin{cases} 0 & \text{if } \frac{\partial \boldsymbol{M}_{y^*}(\mathbf{x}')}{\partial x'_i} < 0 \text{ or } \sum_{j \neq y^*} \frac{\partial \boldsymbol{M}_j(\mathbf{x}')}{\partial x'_i} > 0 \\ \frac{\partial \boldsymbol{M}_{y^*}(\mathbf{x}')}{\partial x'_i} \cdot \mid \sum_{j \neq y^*} \frac{\partial \boldsymbol{M}_j(\mathbf{x}')}{\partial x'_i} \mid & \text{otherwise} \end{cases} \tag{17}$$

- JSMA^{-}:

$$s_{y^*}(x'_i) = \begin{cases} 0 & \text{if } \frac{\partial \boldsymbol{M}_{y^*}(\mathbf{x}')}{\partial x'_i} > 0 \text{ or } \sum_{j \neq y^*} \frac{\partial \boldsymbol{M}_j(\mathbf{x}')}{\partial x'_i} < 0 \\ \mid \frac{\partial \boldsymbol{M}_{y^*}(\mathbf{x}')}{\partial x'_i} \mid \cdot (\sum_{j \neq y^*} \frac{\partial \boldsymbol{M}_j(\mathbf{x}')}{\partial x'_i}) & \text{otherwise} \end{cases} \tag{18}$$

- COI:

$$s_{y^*}(x'_i) = x'_i \cdot \frac{\partial \boldsymbol{M}_{y^*}(\mathbf{x}')}{\partial x'_i} \tag{19}$$

5 Experiments

This experiment compares `pbATN` with other state-of-the-art methods including targeted FGSM, box-constrained L-BFGS, and Carnili-Wagner L_2. These methods are well-known methods to generate adversarial examples by using L_p-norm distance metrics. We implement targeted FGSM and box-constrained L-BFGS because the source code is not available. For Carnili-Wagner L_2, we use the existing source code[1]. The experiment is performed on Google Colab. To demonstrate the advantages of `pbATN`, the experiment addresses the following questions:

- **Improvement in Diversity Issue**: Can the proposed $\texttt{pbATN}_G$ generate diverse adversarial examples in comparison with targeted FGSM, box-constrained L-BFGS, and Carnili-Wagner L_2? (denoted by RQ1)
- **Improvement in Quality Issue**: Does the proposed $\texttt{pbATN}_I$ improve the quality of adversarial examples effectively? (denoted by RQ2)
- **Overall Performance Analysis**: Does the proposed `pbATN` achieve good performance compared to targeted FGSM, box-constrained L-BFGS, and Carnili-Wagner L_2? (denoted by RQ3)

[1] https://github.com/carlini/nn_robust_attacks

5.1 Dataset and Attacked Models

Dataset: This experiment conducts on the MNIST dataset (Lecun et al., 1998b) and CIFAR-10 (Krizhevsky et al.) dataset to demonstrate the advantages of `pbATN`. These two datasets are used commonly to evaluate the robustness of CNN models (Zhang et al., 2019). The MNIST dataset is a collection of handwritten digits. This dataset has 60,000 images on the training set and 10,000 images of size 28 × 28 × 1 on the test set. There are 10 labels representing digits from 0 to 9. The CIFAR-10 dataset is a collection of images for general-purpose image classification. This dataset consists of 50,000 images on the training set and 10,000 images of size 32 × 32 × 3 on the test set. Each pixel is represented in a combination of three colors including red, green, and blue. There are 10 labels such as airplanes, cars, cats, birds, etc.

The result of attacks $m \rightarrow n$ is stacked convolutional autoencoders. These autoencoders reconstruct attacking images labelled m into adversarial examples labelled n. To evaluate attacks $m \rightarrow n$, the experiments split the MNIST dataset and the CIFAR-10 dataset into separate subsets for evaluation. Let $\mathcal{M}_i$ be a subset of the MNIST dataset and $\mathcal{C}_i$ be a subset of the CIFAR-10 dataset. For simplicity, each subset of these datasets is called dataset. Based on the purpose of the evaluation, the experiments define three groups of datasets, namely G_1, G_2, and G_3. Group $G_1 = \{\mathcal{M}_1, \mathcal{C}_1\}$ is a portion of the training sets and used to train autoencoders. Each dataset of G_1 consists of 1,000 attacking images labelled as m. Group $G_2 = \{\mathcal{M}_2, \mathcal{C}_2\}$ is a portion of the training sets and not used to train autoencoders. Each dataset of G_2 contains 4,000 attacking images labelled as m. Group $G_3 = \{\mathcal{M}_3, \mathcal{C}_3\}$ is the test sets of the attacked models. $\mathcal{M}_3$ has 1,000 attacking images labelled as m. Due to the accuracy limitation of two CNNs on CIFAR-10 test set, we set $\mathcal{C}_3$ to have 862 and 759 attacking images labelled as m for AlexNet and LeNet-5, respectively. Group G_2 and Group G_3 are used to evaluate the generalization ability of the autoencoders trained on Group G_1.

Attacked Models: This research uses LeNet-5 (Lecun et al., 1998a) and AlexNet (Krizhevsky et al., 2017) to train on the MNIST dataset and the CIFAR-10 dataset. LeNet-5 is proposed to recognize handwritten

digits such as the MNIST dataset. AlexNet is introduced to deal with general-purpose image classification such as the CIFAR-10 dataset. This architecture is considered one of the most influential CNN architectures. The accuracies of the trained models using these architectures are shown in Table 1. As can be seen, on the MNIST dataset, LeNet-5 produces better results than AlexNet. On the CIFAR-10 dataset, AlexNet outperforms LeNet-5.

Table 1: The accuracy of attacked models on the training set and the test set

Dataset	Model	Training set	Test set
MNIST	AlexNet	99.03%	98.75%
	LeNet-5	99.86%	98.82%
CIFAR-10	AlexNet	99.70%	76.22%
	LeNet-5	97.71%	69.40%

The purpose is to attack these models to generate diverse and high-quality adversarial examples. For each CNN model, there are 90 attacks in total (i.e., 10 original labels $\times$ 9 target labels). However, in this research, we selects some representative attacks to make a comprehensive comparison. Specifically, on the MNIST dataset, the attack is $9 \rightarrow 7$ for AlexNet and $9 \rightarrow 4$ for LeNet-5. On the CIFAR-10 dataset, the attack is *truck* $\rightarrow$ *horse* for AlexNet and *truck* $\rightarrow$ *deer* for LeNet-5.

5.2 Configuration

5.2.1 $\texttt{pbATN}_{G}$

Stacked Convolutional Autoencoder: It is difficult to select an architecture of stacked convolutional autoencoder which attacks well on a CNN model. Instead, the architecture of the stacked convolutional autoencoder should be chosen manually based on the experience of attackers. In this research, for each dataset, the experiments use a stacked convolutional autoencoder for all attacked models. These stacked convolutional autoencoders are shown in Table 2 and Table 3.

Group G_1 is used to train autoencoders. Each autoencoder is trained up to 500 epochs with the batch size of 256. The experiment uses *early stopping* strategy to terminate the training when there is no improvement in the loss over subsequent epochs. After the training process, attacking images would be fed into the trained autoencoders. The purpose of this step is to generate candidate adversarial examples. After that, candidate adversarial examples are predicted by the attacked model to identify if they are valid. If a candidate adversarial example is classified as n, it is an adversarial example.

Table 2: The architecture of the stacked convolutional autoencoder used to generate adversarial examples from dataset $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ of the MNIST dataset

Layer Type	Input	Output Shape
InputLayer	(28, 28, 1)	(28, 28, 1)
Conv2D	(28, 28, 1)	(28, 28, 32)
MaxPooling2D	(28, 28, 32)	(14, 14, 32)
Conv2D	(14, 14, 32)	(14, 14, 32)
MaxPooling2	(14, 14, 32)	(7, 7, 32)
Conv2D	(7, 7, 32)	(7, 7, 32)
UpSampling2D	(7, 7, 32)	(14, 14, 32)
Conv2D	(14, 14, 32)	(14, 14, 32)
UpSampling2	(14, 14, 32)	(28, 28, 32)
Conv2D	(28, 28, 32)	(28, 28, 1)

Table 3: The architecture of the stacked convolutional autoencoder used to generate adversarial examples from dataset $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ of the CIFAR-10 dataset

Layer Type	Input	Output Shape
InputLayer	(32, 32, 3)	(32, 32, 3)
Conv2D	(32, 32, 3)	(32, 32, 32)
MaxPooling2D	(32, 32, 32)	(16, 16, 32)
Conv2D	(16, 16, 32)	(16, 16, 32)
MaxPooling2	(16, 16, 32)	(8, 8, 32)
Conv2D	(8, 8, 32)	(8, 8, 32)
UpSampling2D	(8, 8, 32)	(16, 16, 32)
Conv2D	(16, 16, 32)	(16, 16, 32)
UpSampling2	(16, 16, 32)	(32, 32, 32)
Conv2D	(32, 32, 32)	(32, 32, 3)

The value of weight ϵ: We need to fine-tune weight ϵ in Equation 12 to generate good adversarial examples. This equation consists of two terms called distance term and cross-entropy term. To reduce the effort of choosing such weights, all attacks are performed with a set of fixed weights $\{0.01, 0.05, 0.5, 0.95, 0.99, 1\}$. If ϵ equals 0.01 or 0.05, the attacks tend to minimize the distance term. If ϵ equals 0.5, the attacks minimize the two terms with equal weights. If ϵ equals 0.95 or 0.99, the attacks tend to minimize the cross-entropy term. If ϵ equals 1, the attacks only minimize the cross-entropy term.

Pattern: The experiments use three different patterns to generate adversarial examples including *border pattern*, *class model map pattern*, and *all-feature pattern*. For *class model map pattern*, this pattern describes the impact of features on the classification of the attacked models.

On the MNIST dataset, we empirically choose the top-100 features on the class model map. The attacks would add perturbations to 100 fixed features to generate adversarial examples. On the CIFAR-10 dataset, we empirically choose the top-400 features on the class model map because the number of features on this dataset is about four times larger than that of the MNIST dataset.

5.2.2 $\texttt{pbATN}_I$

$\texttt{pbATN}_I$ is used to improve the quality of adversarial examples in terms of L_0-norm and L_2-norm. $\texttt{pbATN}_I$ uses four main parameters including step α, threshold δ, decay rate t, and ranking heuristics. These parameters are used to adjust the performance of $\texttt{pbATN}_I$. In our experiment, the value of δ is assigned to 0. The value of decay rate t is assigned to 2.

Step α: By analyzing the result of different steps α, we observed that using the appropriate value of step α would have a positive impact on the performance of $\texttt{pbATN}_I$. Using a too large value of step α or a too small value of step α might slow down the performance of $\texttt{pbATN}_I$. Based on the empirical data, we suggest using step $\alpha = 6$ for attacking the MNIST dataset and step $\alpha = 30$ for attacking the CIFAR-10 dataset.

Ranking heuristics: The experiment uses three ranking heuristics including COI, JSMA, and random. Heuristics random is used as a baseline. While heuristics COI and heuristics JSMA arrange adversarial features in increasing order of impact on the target neuron, heuristics random sorts adversarial features randomly.

By analyzing the result of ranking heuristics, for the MNIST dataset, applying heuristics COI or heuristics JSMA tend to converge the average restored rate faster than using heuristics random. For the CIFAR-10 dataset, we observed that using JSMA usually improve adversarial examples most effectively.

5.2.3 Compared methods

We compare pbATN with other state-of-the-art methods including targeted FGSM, box-constrained L-BFGS, and Carnili-Wagner L_2. The value of weights in targeted FGSM and box-constrained L-BFGS is the same as $\texttt{pbATN}_G$. Concerning Carnili-Wagner L_2, this method does not need to choose weight manually. This method uses a binary search algorithm to find the optimal weight to produce high-quality adversarial with a high success rate.

5.3 Answers for Research Questions

5.3.1 RQ1: Improvement in Diversity Issue

This section investigates the ability of $\texttt{pbATN}_G$ to improve the low diversity of adversarial examples. In order to generate diverse adversarial examples, the experiment applies different patterns including *border pattern*, *class model map pattern*, and *all-feature pattern*. To measure the ability to mitigate the diverse issue, the experiment uses the average success rate metric. The average success rate is the average of success rates resulting from the usage of the configured weights. The conclusion is that $\texttt{pbATN}_G$ could achieve good average success rates compared to the other methods.

Table 4 compares the average success rates of the $\texttt{pbATN}_G$ and other comparable methods. There are three promising results. The first result is that *all-feature pattern* could achieve the best average success rates among the used patterns for the MNIST dataset and the CIFAR-10 dataset. In most cases, the average success rate of this pattern is above 99%. The major reason is that rather than changing a subset of features as *border pattern* and *class model map pattern*, this pattern allows adding adversarial perturbation to all features. The autoencoders could learn characteristics hidden in all features of attacking images to generate adversarial examples. As a result, the features with more impact on the classification tend to be modified by the trained autoencoders more than the features with lower impact. Concerning the other patterns, *border pattern* tends to work better than *class model map pattern* for the attacking images on the MNIST dataset. In particular, while *border pattern* achieves from 17.1% to 55.1%, *class model map pattern* achieves

Table 4: The average success rate comparison (%)

Model	Dataset	pbATN$_G$			Carnili-Wagner L_2	Targeted FGSM	L-BFGS
		Border pattern	*Class model pattern*	*All-feature pattern (ATN)*			
AlexNet $9 \rightarrow 7$	$\mathcal{M}_1$	21.2	14.9	99.9	100	8.2	23.1
	$\mathcal{M}_2$	17.1	22.2	99.4	100	8.1	21.8
	$\mathcal{M}_3$	20.8	9.4	99.1	99.2	6.7	23.4
LeNet-5 $9 \rightarrow 4$	$\mathcal{M}_1$	55.1	23.2	95.5	100	20.1	34.2
	$\mathcal{M}_2$	45.8	23.8	99.2	100	19.9	36.4
	$\mathcal{M}_3$	52.3	20.6	92.2	99.1	22.4	33.1
AlexNet *truck* $\rightarrow$ *horse*	$\mathcal{C}_1$	8.4	68.3	100	100	5.4	24
	$\mathcal{C}_2$	6.2	62.6	99.2	99.1	5.6	22.4
	$\mathcal{C}_3$	6.5	67.5	98.1	99.3	3.9	23.7
LeNet-5 *truck* $\rightarrow$ *deer*	$\mathcal{C}_1$	28.3	61.1	100	100	18.7	36.7
	$\mathcal{C}_2$	24.1	54.3	99.3	100	20.7	34.6
	$\mathcal{C}_3$	22.6	58.5	99.1	98.1	15.8	38.3

from 9.4% to 23.8%. Otherwise, for the attacking images on the CIFAR-10 dataset, *class model map pattern* produces better results with the average success rate from about 54.3% to 68.3%, than *border pattern* with the average success rate from 6.2% to 28.3%.

The second result is that the average success rates of *all-feature pattern* in pbATN$_G$ are better than box-constrained L-BFGS, targeted FGSM, and approximate to Carnili-Wagner L_2. Targeted FGSM has the worst average success rate because this method only modifies attacking images in one iteration to generate adversarial examples. Box-constrained L-BFGS seems to be better than targeted FGSM because they use gradient descent to generate adversarial examples. However, targeted FGSM and box-constrained L-BFGS are sensitive to the values of the weight in Equation 6 and Equation 7, which are selected based on the experience of the testers. Concerning Carnili-Wagner L_2, this method usually generates adversarial examples with 100% average success rate, which was shown in Carlini and Wagner (2016). The main reason is that the optimal weight c in Equation 8 is found out automatically by applying a binary search algorithm. It is promising that the average success rate of pbATN$_G$ is mostly 99%-100%, which is approximate to Carnili-Wagner L_2.

The third result is that pbATN$_G$ could generate adversarial examples from new datasets with the average success rate approximate to the trained datasets. The trained datasets are in Group G_1 (i.e., $\mathcal{M}_1$ and $\mathcal{C}_1$). The new datasets are in Group G_2 (i.e., $\mathcal{M}_2$ and $\mathcal{C}_2$) and Group G_3 (i.e., $\mathcal{M}_3$ and $\mathcal{M}_3$). The new datasets have the same classification purpose as the trained datasets. The maximum difference between the average success rate on the new datasets and the trained datasets is about 10%, which is acceptable. It means that if the trained autoencoders have a high average success rate on the trained datasets, these autoencoders would likely achieve approximate average success rates on other similar datasets. This is the generalization ability of pbATN$_G$, which is not supported in the comparable methods.

5.3.2 RQ2: Improvement in Quality Issue

This section investigates the ability of the proposed pbATN$_I$ to enhance the quality of adversarial examples. The conclusion is that pbATN$_I$ could enhance the quality of adversarial examples significantly for pbATN$_G$, targeted FGSM, and box-constrained L-BFGS. For the adversarial examples generated by Carnili-Wagner L_2, pbATN$_I$ could enhance their quality slightly.

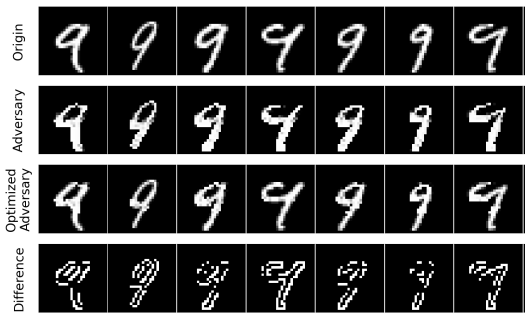
In pbATN$_I$, rather than improving the quality of adversarial examples individually, we put them in a batch and improve at the same time to reduce the computational cost. Because the used heuristics produce approximate final results, the experiment reports the average reduction rate for simplicity. The average reduction rate is computed by $(a - b)/a$, where a is the average L_p-norm distance without applying the ranking heuristics and b is the average L_p-norm distance after applying the ranking heuristics.

Table 5: The average reduction rate (%) of adversarial examples by applying $\texttt{pbATN}_I$. Larger values are better.

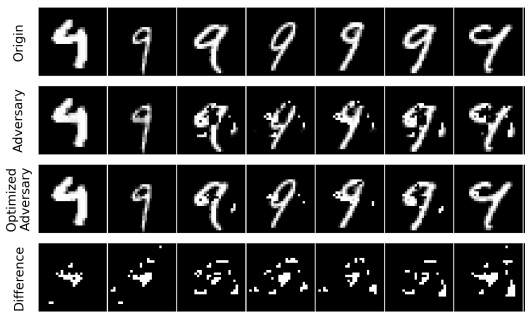
Model	Method	L_0	L_2
AlexNet 9 → 7	$\texttt{pbATN}_G$+ *border*	75.9	42.9
	$\texttt{pbATN}_G$+ *class model map*	84.4	51.0
	$\texttt{pbATN}_G$+ *all-feature* (ATN)	98.0	69.6
	Targeted FGSM	91.5	69.2
	Box-constrained L-BFGS	93.4	75.6
	Carnili-Wagner L_2	7.9	0.1
LeNet-5 9 → 4	$\texttt{pbATN}_G$+ *border*	70.1	38.9
	$\texttt{pbATN}_G$+ *class model map*	71.9	36.9
	$\texttt{pbATN}_G$+ *all-feature* (ATN)	96.7	65.2
	Targeted FGSM	85.7	68.5
	Box-constrained L-BFGS	88.6	56.8
	Carnili-Wagner L_2	6.5	0.1
AlexNet *truck* → horse	$\texttt{pbATN}_G$+ *border*	96.3	78.5
	$\texttt{pbATN}_G$+ *class model map*	85.8	60.2
	$\texttt{pbATN}_G$+ *all-feature* (ATN)	89.7	67.6
	Targeted FGSM	87.8	82.7
	Box-constrained L-BFGS	95.7	79.0
	Carnili-Wagner L_2	0.8	4.5
LeNet-5 *truck* → deer	$\texttt{pbATN}_G$+ *border*	95.7	78.0
	$\texttt{pbATN}_G$+ *class model map*	88.4	62.4
	$\texttt{pbATN}_G$+ *all-feature* (ATN)	89.4	65.9
	Targeted FGSM	98.0	89.0
	Box-constrained L-BFGS	94.0	82.9
	Carnili-Wagner L_2	3.3	1.3

Table 5 shows the average reduction rate of L_0-norm and L_2-norm when applying $\texttt{pbATN}_I$. Generally, L_0-norm and L_2-norm could be reduced by applying $\texttt{pbATN}_I$. Especially, concerning L_0-norm, we observed that $\texttt{pbATN}_I$ could decrease from a large number of adversarial features to one adversarial feature. For $\texttt{pbATN}_G$, targeted FGSM, and box-constrained L-BFGS, $\texttt{pbATN}_I$ could enhance the quality of the generated adversarial examples significantly. $\texttt{pbATN}_I$ could decrease at most about 98% L_0-norm and 89% L_2-norm. For Carnili-Wagner L_2, $\texttt{pbATN}_I$ could improve the quality of adversarial examples slightly. This is explainable since Carnili-Wagner L_2 usually generates adversarial examples with a minimum L_2-norm. This method uses a binary search algorithm to find out the optimal weight in Equation 8, which leads to a minimum L_2-norm.

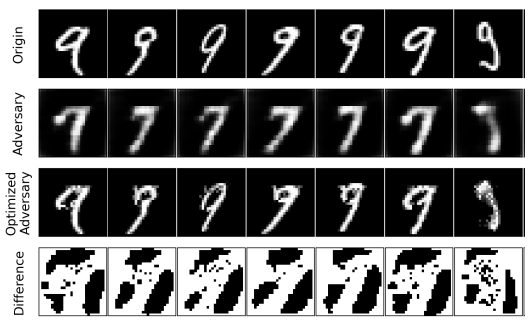
Examples of some improvements for adversarial examples are shown in Fig. 6. Notions *Origin* and *Adversary* denote the attacking images and their corresponding adversarial examples, respectively. Notion *Difference* shows the difference between adversarial examples and their improvement in white colour.



(a) *Border pattern*



(b) *Class model map pattern*



(c) *All-feature pattern*

Fig. 6: Examples of improved adversarial examples generated by $\texttt{pbATN}_I$.

5.3.3 RQ3: Overall Performance Analysis

Table 6 shows the overall performance of pbATN and other comparable methods. pbATN includes $\texttt{pbATN}_G$ and $\texttt{pbATN}_I$. The comparable unit is second. Generally, pbATN could achieve good average success rates while requiring lower computational

Table 6: The overall performance comparison (seconds) between `pbATN` and other methods

Model	Dataset	`pbATN`			Carnili-Wagner L_2	Targeted FGSM	L-BFGS
		Border pattern	*Class model pattern*	*All-feature pattern (ATN)*			
AlexNet $9 \rightarrow 7$	$\mathcal{M}_1$	98	131	174	1,815	18	240
	$\mathcal{M}_2$	21	38	87	7,176	61	1,009
	$\mathcal{M}_3$	11	21	63	1,716	20	258
LeNet-5 $9 \rightarrow 4$	$\mathcal{M}_1$	84	109	95	714	11	141
	$\mathcal{M}_2$	13	9	55	2,597	40	549
	$\mathcal{M}_3$	6	4	46	728	12	128
AlexNet *truck* $\rightarrow$ *horse*	$\mathcal{C}_1$	216	287	309	2,255	18	289
	$\mathcal{C}_2$	26	55	125	8,942	68	1,152
	$\mathcal{C}_3$	13	24	99	2,164	17	324
LeNet-5 *truck* $\rightarrow$ *deer*	$\mathcal{C}_1$	173	181	190	917	42	158
	$\mathcal{C}_2$	13	44	53	3,276	195	622
	$\mathcal{C}_3$	8	21	47	919	40	148

cost considerably compared to the other methods. This is a promising result to show that `pbATN` could be applied in practice.

We need to evaluate the overall performance of `pbATN` when dealing with new datasets. For all patterns, the experiment shows that `pbATN` could generate adversarial examples from new datasets (i.e., Group G_2 and Group G_3) with lower computational cost than the trained datasets (i.e., Group G_1). The computational cost on Group G_1 includes the training process, the adversarial example generation process, and the improvement process for adversarial examples. The computation cost on Group G_2 and Group G_3 includes the adversarial example generation process and the improvement process for adversarial examples. As can be seen, the overall performance of adversarial example generation on Group G_2 and Group G_3 is from 2 times to 27 times faster than that of Group G_1. It means that the machine learning testers only need to spend effort on training autoencoders. The trained autoencoders could be reused for generating adversarial examples from new attacking images with an extremely low computational cost. By contrast, the other methods does not support generating adversarial examples based on the information of previous adversarial example generation. In other words, every adversarial example is generated independently to each other. As a result, when dealing with a set of attacking images, `pbATN` usually produces adversarial examples faster than the other methods.

Another promising finding is that `pbATN` could mitigate the trade-off between the average success rate and the overall performance. Targeted FGSM and box-constrained L-BFGS are mostly worse than `pbATN` in terms of the average success rate and overall performance. About Carnili-Wagner L_2, although this method could achieve mostly 100% average success rate as shown in Table 4, this method consumes a large amount of computational cost, which is mostly from about 1,000 seconds to around 3,000 seconds. By contrast, by appling *all-feature pattern*, `pbATN` could not only achieve approximate average success rates as Carnili-Wagner L_2 but also require a smaller computational cost significantly. In particular, the computational cost of applying *all-feature pattern* is from 7.5 times (i.e., LeNet-5 with dataset $\mathcal{M}_1$) to 80 times (i.e., AlexNet with dataset $\mathcal{M}_2$) faster than Carnili-Wagner L_2.

6 Discussion

Integral constraint: On the MNIST dataset and the CIFAR-10 dataset, the value of each feature is an integer in the set $D = \{0, 1, ..., 255\}$. After normalization, the value of each feature is in the set $D' = \left\{0, \frac{1}{255}, \frac{2}{255}, ..., 1\right\}$. The value of an adversarial feature must be in the valid set D or D'. However, similarly to targeted FGSM (Goodfellow et al., 2015) and Carnili-Wagner (Carlini and Wagner, 2016), we ignore this requirement in the objective function of the stacked convolutional autoencoder in the experiments. Instead, the output of the stacked convolutional autoencoder is

continuous in the range [0, 1]. After that, this output of the stacked convolutional autoencoder is normalized to a valid domain. The output $\mathbf{x}'$ of the stacked convolutional autoencoder is an adversarial example if and only if $\boldsymbol{M}\left(\frac{\lfloor \mathbf{x}' * 255 \rfloor}{255}\right) = y^*$, where y^* is the target label. We see that ignoring the integral constraint of adversarial examples in the objective of the stacked convolutional autoencoder rarely affects the success rates of the attacks.

The generation of class model map: A class model map $\mathcal{I}$ of a label l is generated from a set of attacking images classified as l. In the Objective 14, the process of finding $\mathcal{I}$ is a training procedure. A major problem of this procedure is that the generated $\mathcal{I}$ could be a locally optimal class model map. This issue could be addressed by applying different gradient descent such as Backpropagation (Springenberg et al., 2014), Excitation Backprop (Zhang et al., 2016), etc.

7 Conclusion

We have presented `pbATN` method to generate high-quality and diverse adversarial examples. `pbATN` includes two phases named $\texttt{pbATN}_G$ and $\texttt{pbATN}_I$. Firstly, we propose $\texttt{pbATN}_G$, which is a pattern-based stacked convolutional autoencoder, to generalize ATN. This autoencoder is used to generate diverse adversarial examples for CNN models. We conducted experiments with three patterns including *border pattern*, *all-feature pattern*, and *class model map pattern* in $\texttt{pbATN}_G$. Secondly, we introduce $\texttt{pbATN}_I$ to enhance the quality of adversarial examples. The main idea of $\texttt{pbATN}_I$ is to restore *redundantly adversarial features* based on their impacts on the classification of the attacked model.

The comprehensive experiment on the MNIST dataset and the CIFAR-10 dataset has shown several promising results. In terms of adversarial diversity, $\texttt{pbATN}_G$ could generate diverse adversarial examples by using different patterns. Especially, for *all-feature pattern*, most of the attacks could achieve above 99% average success rate. In terms of adversarial quality, concerning L_0-norm, $\texttt{pbATN}_I$ could reduce from hundreds of adversarial features to one adversarial feature. About L_2-norm, our $\texttt{pbATN}_I$ could reduce the average distance considerably. $\texttt{pbATN}_I$ could enhance the quality of adversarial examples generated by targeted FGSM and box-constrained L-BFGS significantly and Carnili-Wagner L_2 slightly. Additionally, we suggest using the adversarial feature ranking heuristics COI and JSMA in $\texttt{pbATN}_I$ to achieve better performance than using the random heuristics. In the future, `pbATN` will be extended to support other distance metrics such as L_∞-norm. In addition, the research will make a more comprehensive comparison with other well-known datasets.

8 Declaration

8.1 Authorship Contributions

Conceptualization: Pham Ngoc Hung, Duc-Anh Nguyen
Methodology: Duc-Anh Nguyen, Kha Do Minh
Formal analysis and investigation: Duc-Anh Nguyen, Kha Do Minh
Writing - original draft preparation: Duc-Anh Nguyen
Writing - review and editing: all authors

8.2 Compliance with Ethical Standards

8.2.1 Ethics approval

Not applicable.

8.2.2 Funding

The authors did not receive support from any organization for the submitted work.

8.2.3 Conflicts of interest

The authors have no relevant financial or non-financial interests to disclose.

8.2.4 Informed Consent

Not applicable

8.3 Availability of data and material

All data and material are available.

8.4 Code availability

All source code are available.

References

Naveed Akhtar, Ajmal Mian, Navid Kardan, and Mubarak Shah. Threat of adversarial attacks on deep learning in computer vision: Survey ii, 2021.

Michael A. Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh Nguyen. Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. *CoRR*, abs/1811.11553, 2018.

Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples, 2017.

Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. *CoRR*, abs/1605.07262, 2016.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, page 153–160, Cambridge, MA, USA, 2006. MIT Press.

Tom B. Brown, Nicholas Carlini, Chiyuan Zhang, Catherine Olsson, Paul Christiano, and Ian Goodfellow. Unrestricted adversarial examples, 2018.

Chunshui Cao, Xianming Liu, Yi Yang, Yinan Yu, Jiang Wang, Zilei Wang, Yongzhen Huang, Liang Wang, Chang Huang, Wei Xu, Deva Ramanan, and Thomas S. Huang. Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks. pages 2956–2964, 2015.

Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016.

Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers, 2017.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Discovering adversarial examples with momentum. *CoRR*, abs/1710.06081, 2017.

Christian Etmann, Sebastian Lunz, Peter Maass, and Carola-Bibiane Schönlieb. On the connection between adversarial robustness and saliency map interpretability, 2019.

Ruth Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. *CoRR*, abs/1704.03296, 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. 2015.

Divya Gopinath, Corina S. Păsăreanu, Kaiyuan Wang, Mengshi Zhang, and Sarfraz Khurshid. Symbolic execution for attribution and attack synthesis in neural networks. IEEE Press, 2019.

Jindong Gu and Volker Tresp. Saliency methods for explaining adversarial attacks. *CoRR*, abs/1908.08413, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

IEEE. Ieee standard glossary of software engineering terminology, 1990.

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 2014.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. 60(6), 2017. ISSN 0001-0782.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998a. doi: 10.1109/5.726791.

Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998b.

Lei Ma, Felix Juefei-Xu, Jiyuan Sun, Chunyang Chen, Ting Su, Fuyuan Zhang, Minhui Xue,

Bo Li, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. Deepgauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems. *CoRR*, abs/1803.07519, 2018.

Ravi Mangal, Aditya V. Nori, and Alessandro Orso. Robustness of neural networks: A probabilistic and practical approach. *CoRR*, abs/1902.05983, 2019.

Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski, editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, pages 52–59, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. *CoRR*, abs/1511.04599, 2015.

Hanieh Naderi, Leili Goli, and Shohreh Kasaei. Generating unrestricted adversarial examples via three parameters. *CoRR*, abs/2103.07640, 2021.

Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 2011.

Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528, 2015.

Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. *CoRR*, abs/1705.06640, 2017.

Yunchen Pu, Weiyao Wang, Ricardo Henao, Liqun Chen, Zhe Gan, Chunyuan Li, and Lawrence Carin. Adversarial symmetric variational autoencoder. *CoRR*, abs/1711.04915, 2017.

K. Simonyan, A. Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *CoRR*, abs/1710.08864, 2017.

Farhana Sultana, Abu Sufian, and Paramartha Dutta. Advancements in image classification using convolutional neural network. *CoRR*, abs/1905.03288, 2019.

Youcheng Sun, Xiaowei Huang, and Daniel Kroening. Testing deep neural networks. *CoRR*, abs/1803.04792, 2018a.

Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. *CoRR*, abs/1805.00089, 2018b.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 2014.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein autoencoders, 2019.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy, 2019.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010. ISSN 1532-4435.

Fuxun Yu, Qide Dong, and Xiang Chen. ASP: A fast adversarial attack example generation framework based on adversarial saliency prediction. *CoRR*, abs/1802.05763, 2018.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks, 2013.

Jianming Zhang, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *CoRR*, abs/1608.00507, 2016.

Jie Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. 06 2019.