DATA ANALYTICS AND MACHINE LEARNING

Improving diversity and quality of adversarial examples in adversarial transformation network

Duc-Anh Nguyen¹ · Kha Do Minh¹ · Khoi Nguyen Le¹ · Le-Minh Nguyen² · Pham Ngoc Hung¹

Accepted: 14 November 2022 / Published online: 29 November 2022
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract
This paper proposes PatternAttack to mitigate two major issues of Adversarial Transformation Network (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 patterns to generalize ATN. This proposed autoencoder could support different patterns such as all-pixel pattern, object boundary 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 $L_0$-norm and $L_2$-norm. This algorithm employs adversarial pixel ranking heuristics such as JSMA and COI to prioritize adversarial pixels. 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 generates diverse adversarial examples. For the second issue, the proposed algorithm significantly improves the quality of adversarial examples. In terms of $L_0$-norm, the proposed algorithm decreases from hundreds of adversarial pixels to one adversarial pixel. In terms of $L_2$-norm, the proposed algorithm reduces the average distance considerably. These results show that the proposed method can generate high-quality and diverse adversarial examples in practice.

Keywords Adversarial example · Robustness · Stacked convolutional autoencoder · Targeted attack

1 Introduction

Convolutional Neural Network (CNN) is usually applied to classify images (Sultana et al. 2019). CNN is trained to learn important pixels on a labelled dataset consisting of images and their labels to predict the label of new images. CNN could achieve high accuracy on the training set and the test set. However, in practice, the accuracy could be significantly lower than expected (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 CNN in terms of accuracy, precision, or F1-score. Meanwhile, in real-world situations, input data could contain perturbation, which can be rarely existed in the training set and the test set. Attackers could create perturbational inputs to interfere the trained models to behave unexpectedly. Therefore, it is important to evaluate the behaviors of CNN in the presence of perturbation.

Robustness is one of the popular measurements to evaluate the quality of CNN in the presence of perturbation (IEEE 1990; Carlini and Wagner 2016; Baluja and Fischer 2017; Zhang et al. 2019). To ensure the robustness of CNN, an adversarial attack is a popular approach which attempts to modify a correctly predicted image. There are two main types of this approach including untargeted attack and targeted attack (Akhtar et al. 2021). These two types of attacks aim to modify a correctly predicted input intentionally. 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, an autoencoder-based attack is a promising approach to generate adversarial examples. This approach was firstly proposed in Adversarial Transformation Network (ATN) (Baluja and Fischer 2017). The general idea is that an autoencoder is trained from an attacked model, a set of correctly predicted inputs, and a target label. The main advantage of ATN is that the trained autoencoder could be reused for generating adversarial examples from new correctly predicted inputs 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 low-diversity adversarial examples. This method modifies all pixels to produce adversarial examples from correctly predicted inputs. However, some specific regions of correctly predicted inputs 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 correctly predicted inputs 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 correctly predicted inputs to generate adversarial examples.

Secondly, ATN usually generates low-quality adversarial examples. The purpose of ATN is to generate adversarial examples close to the correctly predicted inputs. Their differences are measured by using \( L_2 \)-norm. These distances are small if the adversarial examples are close to their corresponding correctly predicted inputs. A stacked convolutional autoencoder can be utilized to generate adversarial examples from the correctly predicted inputs. 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 correctly predicted inputs. In the worst case, perturbations are added to all pixels of correctly predicted inputs. As a result, the corresponding adversarial examples could be very different from their corresponding correctly predicted inputs. Therefore, it is worthwhile to improve the quality of adversarial examples generated by ATN.

This paper proposes a method named Pattern-based Adversarial Attack (PatternAttack) to address the two issues. Firstly, to deal with the low diversity of adversarial examples, a generalized ATN is proposed in PatternAttack. Similarly to ATN, the generated adversarial examples should be close to the correctly predicted inputs in terms of \( L_2 \)-norm. The generalized ATN aims to modify a set of specific pixels, which is defined by a pattern. This research suggests several patterns including all-pixel pattern, object boundary pattern, and class model map pattern. Among these patterns, all-pixel pattern is used in ATN. Secondly, to improve the quality of adversarial examples, an optimizer algorithm is proposed in PatternAttack. The proposed optimizer is able to improve the quality of adversarial examples in terms of both \( L_0 \)-norm and \( L_2 \)-norm. The main idea of the proposed optimizer is to restore redundant adversarial pixels in an adversarial example to the original values in the corresponding correctly predicted input. An adversarial pixel 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 pixels 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 correctly predicted input could be reduced significantly.

Addressing the two problems of ATN bring three significant benefits. Firstly, using different patterns in the generalized ATN could generate more evidence about the robustness of CNN. For example, if a machine learning test wants to check whether the object boundary pixels are prone to adversarial attack, they could use the object boundary pattern. The proposal suggests three types of patterns, however, the tester could define additional patterns. Secondly, the proposed optimizer could be used to improve the quality of any adversarial attacks. Currently, many adversarial attacks are proposed to generate high-quality adversarial examples. In many cases, their generated outputs are not as good as expected. By using the proposed optimizer, the problem of choosing an adversarial attack is not much important. For example, the tester could use FGSM (Goodfellow et al. 2015) to attack, then use the proposed optimizer to enhance the output of FGSM. Thirdly, generating various adversarial examples could play an important role to build an adversarial defense. One of the most common approaches is to build a defense and put it in front of CNN (Aldahdooh et al. 2021). Whenever there is a new image sent to CNN (i.e., adversarial or clean), it is sent to the defense to (i) transform to the valid image or (ii) check the validity of the input image. For example, DefenseVAE (Li and Ji 2020) analyzes the insight of adversarial examples generated by FGSM, BIM, CW, etc. and builds a variational autoencoder-based defense.

Briefly, the contribution of the paper is described as follows:

- We propose the generalized ATN, a pattern-based adversarial attack for CNN.
- We propose an optimizer to improve the quality of adversarial examples generated by any adversarial attacks.
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 CNN and the targeted attack. The overview of PatternAttack is shown in Sect. 4. Next, Sect. 5 presents the experiment to demonstrate the advantages of PatternAttack. The discussion is presented in Sect. 6. Finally, the conclusion is described in Sect. 7.

2 Related works

This section presents an overview of related research. Firstly, some well-known adversarial example generation methods for CNN are presented. Secondly, outstanding research on finding the saliency map of CNN is discussed.

2.1 Adversarial example generation

Many adversarial example generation methods are proposed to evaluate the robustness of CNN. Several popular methods could be referred to box-constrained L-BFGS (Szegedy et al. 2014), DeepFool (Moosavi-Dezfooli et al. 2015), FGSM (Goodfellow et al. 2015), Carlini-Wagner (Carlini and Wagner 2016), BIS (Kurakin et al. 2016), ATN (Baluja and Fischer 2017), MI-FGSM (Dong et al. 2017), DeepCheck (Gopinath et al. 2019), and Nguyen-Pham (Nguyen et al. 2022). Table 1 describes the overview of these adversarial attacks.

Pattern attack shows the pattern of the possible modified pixels. The comparable methods support all-pixel pattern such as FGSM, CW or n-most important pixel pattern such as DeepCheck and Nguyen-Pham. Meanwhile, our proposal could modify a set of pixels satisfying a specific pattern such as all-pixel pattern, object boundary pattern, and class model map pattern. The purpose of using various patterns is to clarify the robustness of the attacked model or create various adversarial inputs for building defenses against adversarial attacks.

Optimization is to enhance the quality of adversarial examples. The comparable methods do not support this capability. These methods usually minimize the distance between the generated adversarial examples and their correctly predicted inputs to achieve high-quality outputs. However, it is hard to generate optimal adversarial examples because the objective function of CNN is nonlinear. By contrast, the proposal could recognize and remove such redundant adversarial perturbations in the generated adversarial example.

For generalization, if one method could generate new adversarial examples from new correctly predicted inputs based on the result of the previous attacks, it has the generalization ability. Only ATN and the proposal support this ability. From the correctly predicted inputs, ATN and the pro-

| Category/Method | L-BFGS | DeepFool | CW | MI-FGSM | ATN | Pattern attack | Optimization | Generalization | Distance metric | Pixel modification | Pixel modification |
|-----------------|--------|----------|----|---------|-----|----------------|-------------|---------------|-----------------|-------------------|-------------------|
| Pattern attack  | All-pixel | No      | No | No      | No  | All-pixel      | No          | No            | No              | No                | No                |
| Optimization    | Generalization | Distance metric | Pixel modification | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent |
| Generalization  | No      | No      | No | No      | No  | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent |
| Distance metric | No      | No      | No | No      | No  | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent |
| Pixel modification | No      | No      | No | No      | No  | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent |
| Pixel modification | No      | No      | No | No      | No  | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent | Gradient descent |
posal would train an autoencoder to convert these images to adversarial examples. Later, the trained autoencoder could be reused to generate new adversarial examples instantly from new inputs.

Distance metric describes the $L_p$-norm that the adversarial attack uses. $L_0$ aims to minimize the number of modified pixels. $L_2$ considers maximizing the Euclidean distance between the generated adversarial examples and their corresponding input images. Using $L_{\infty}$ usually does not lead to better adversarial examples in terms of $L_0$ and $L_2$.

Pixel modification is the mechanism to modify a correctly predicted input. DeepCheck uses one SMT-Solver such as Z3 (Moura and Bjørner 2008), however, the performance of predicted input. DeepCheck uses one SMT-Solver such as Z3 (Moura and Bjørner 2008), however, the performance of predicted input. The other methods use a gradient descent-based technique to modify pixels rather than a solver.

2.2 Saliency map generation

Saliency map generation draws great interest from research groups. Saliency map describes the impact of pixels on the classification of CNN (Simonyan et al. 2013; Gu and Tresp 2019). The saliency map usually has the same size as the input of CNN. Recent research groups have proposed different methods to generate saliency maps.

Simonyan et al. (2013) firstly propose a method to generate saliency map, known as class model map. This map could be used to explain how 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 an adversarial 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 an image-specific class saliency map of an input image. Cao et al. (2015) present a novel feedback 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 CNN. Unlike other saliency map generation methods, their method explicitly modifies the correctly predicted input to generate an adversarial example. The modification of this correctly predicted input 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 correctly predicted input with a low computational cost. Yu et al. (2018) propose a method to predict the 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 robust models have interpretable saliency maps. Etmani et al. (2019) quantify the relationship between the saliency map and correctly predicted input by analyzing their alignment. They conclude that the more linear CNN has a stronger connection between its robustness and alignment.

Due to the existence of the relationship between the saliency map and adversarial example, our research utilizes the class model map (Simonyan et al. 2013) as a pattern in the generalized ATN. By generating a class model map of a correctly predicted input set, the generalized ATN could find top-n important pixels which have the largest impact on the classification of CNN. The generalized ATN then modifies these important pixels to generate adversarial examples.

3 Background

3.1 Convolutional neural network

Definition 1 (Convolutional neural network) A typical convolutional neural network $M$ is a tuple $(L, W, \theta)$, where $L = \{L_k | k \in \{0, ..., h-1\}\}$ 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, $W \subseteq L \times L$ is a set of layers, and $\theta = \{\theta_0, \theta_1, ..., \theta_{h-1}\}$ represents a set of activation functions.

The activation function of the $k$-th layer, denoted by $\theta_k$, 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. Given an input image $x \in \mathbb{R}^n$, a typical CNN $M$ can be illustrated in a simple representation as $M(x) = L_{h-1} \circ L_{h-2} \circ ... \circ L_0 \in \mathbb{R}^k$, where $\circ$ presents linear connection, $k$ is the number of classes, $L_0$ is the input layer, and $L_{h-1}$ is the output layer. The input image has the dimension $(\text{width}, \text{height}, \text{channel})$. If the input image is monochrome, the size of channel is one. If the input image is RGB, the size of channel is three corresponding to red, green, and blue channel.

For example, Fig. 1 illustrates the architecture of LeNet-5 (Lecun et al. 1998a). This CNN model is designed for handwritten and machine-printed character recognition. This model comprises 8 layers. Convolutional layers are labelled $C_i$, where $i$ is the layer index. Sub-sampling layers and fully-connected layers are labelled $S_i$ and $F_i$, respectively. The
dimension of the input image is \((32, 32, 1)\). Layer \(C_0\) is a convolutional layer with 6 feature maps of size \(28 \times 28\). Layer \(S_2\) is a sub-sampling layer with 6 feature maps of size \(14 \times 14\). Layer \(C_3\) is a convolutional layer with 16 feature maps of size \(10 \times 10\). Layer \(S_4\) is a sub-sampling layer with 16 feature maps of size \(5 \times 5\). Layer \(C_5\) is a convolutional layer with 120 feature maps. The size of a feature map in this layer is \(1 \times 1\). Layer \(C_5\) is then flattened into fully-connected layer \(F_0\) with 84 hidden units. The output layer has 10 outputs in which each output is corresponding to a digit.

Let \(X\) be the training set containing a set of images. The probability of \(i\)-th label is denoted by \(M_i(x) \in \mathbb{R}\). Let \(\frac{\partial M_i(x)}{\partial x_i}\) be the gradient of the \(j\)-th output neuron with respect to the pixel \(x_i\). The ground truth label of \(x\) is denoted by \(y_{true}\).

### 3.2 Targeted attack

The key idea of a targeted attack method is that perturbations are added to a correctly predicted input to generate an adversarial example. Let \(x'\) be an adversarial example generated from the attack. This adversarial example is classified as a target label \(y^* \neq y_{true}\). The output neuron corresponding to the target label is called the target neuron.

**Definition 2** (Mandatory perturbation) A perturbation vector \(\xi\) is mandatory if and only if \(x\) is predicted correctly by the attacked model and \((x + \xi)\) produces an adversarial example classified as the target label \(y^*\).

**Definition 3** (Redundant perturbation) A perturbation vector \(\xi\) is redundant if and only if \(x\) and \((x + \xi)\) are classified as the ground true label.

**Definition 4** (Adversarial pixel) Given a correctly predicted input \(x\) and its corresponding adversarial example \(x'\), a pixel \(x'_j\) on the adversarial example \(x'\) is adversarial if and only if \(x'_j \neq x_j\).

**Definition 5** (Redundant adversarial pixel) Given an adversarial example \(x'\), its adversarial pixel \(x'_j\) is redundant if \(x'_j\) contain a redundant perturbation \(\zeta_i\).

Redundant perturbations play an important role in our proposed optimizer. We would identify redundant perturbations and remove them from the adversarial examples. This leads to an improvement in the quality of adversarial examples in terms of \(L_0\)-norm and \(L_2\)-norm. The detail of the proposed optimizer is presented in Sect. 4.2.

### 3.2.1 Distance metric

Given a correctly predicted input \(x\), its adversarial example \(x'\) should be similar to the corresponding correctly predicted input. \(L_p\)-norm is one popular way to measure the difference between a correctly predicted input and its corresponding adversarial example. The equation of \(L_p\)-norm is defined as follows:

\[
L_p(x, x') = \left( \sum_i (x'_i - x_i)^p \right)^{\frac{1}{p}}
\]

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

There is 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 \(L_\infty\)-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), Carlini-Wagner \(L_0\) (Carlini and Wagner 2016), and DeepCheck (Gopinath et al. 2019) minimize \(L_0\)-norm.
3.2.2 Success rate

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

$$sr(M) = \frac{\sum_{x'} \mathbb{1}(\arg\max(M(x')) = y^*)}{|X_{attack}|}$$

(2)

where $X_{attack}$ is a set of correctly predicted inputs, $x'$ is an adversarial example generated by modifying an attacking sample on $X_{attack}$, and $\mathbb{1}$ is an indicator function. Function $\mathbb{1}(\arg\max(M(x')) = y^*)$ returns 1 if the model $M$ classifies $x'$ as the target label $y^*$, and returns 0 otherwise.

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 (Andrew 2011), convolutional (Masci et al. 2011), variational (Kingma and Welling 2014), symmetric (Pu et al. 2017), and Wasserstein (Tolstikhin et al. 2019). 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 the input image and reconstructed image, respectively. This research uses a 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 $x_{in} \in \mathbb{R}^{n \times 1}$. In the encoder stage, this autoencoder maps this input image to a latent representation $z \in \mathbb{R}^{n \times 1}$ in which $n < m$:

$$z = f(W_1^T \cdot x_{in} + b_1)$$

(3)

where $b_1 \in \mathbb{R}^{n \times 1}$ is the bias matrix of the hidden layer, $W_1 \in \mathbb{R}^{m \times n}$ is the weight matrix between the input layer and the hidden layer, and $f(.)$ is an activation function.
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, downsampling, and fully-connected. In the decoder stage, a layer can be deconvolutional, upsampling, and fully-connected. The typical objective function of the stacked convolutional autoencoder is identical to Eq. 5.

Figure 3 illustrates an example of stacked convolutional autoencoder. The input image is a monochrome image with dimension (28, 28, 1). 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 FC, an upsampling layer Reshape and deconvolutional layers DeConv3, DeConv2, and DeConv1 to reconstruct the input image.

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:

\[
\text{minimize } (1 - \beta) \cdot L_2(x', x)^2 + \beta \cdot f(y^*, M(x')) \\
\text{such that } x' \in [0, 1]^n
\]

where \( f \) is a function to compute the difference between \( M(x') \) and the target label \( y^* \), and \( \beta \) 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 correctly predicted inputs in terms of \( L_{\infty} \)-norm. Targeted FGSM generates adversarial examples by modifying all pixels:

\[
x' = x - \beta \cdot \text{sign} \left( \frac{\partial M_{y^*}(x)}{\partial x_i} \right)
\]

where function \( \text{sign}(.) \) returns the sign of \( \frac{\partial M_{y^*}(x)}{\partial x_i} \) and \( \beta \) is a positive value to shift values of all pixels in \( x \).

The main disadvantage of targeted FGSM is that its effectiveness is sensitive to the value of \( \beta \). If the value of \( \beta \) is large, the success rate of the targeted FGSM could be high. However, adversarial examples could be very different from the corresponding correctly predicted inputs in terms of \( L_p \)-norm. Otherwise, if the value \( \beta \) is very small, the success rate of the targeted FGSM could be low. In terms of \( L_p \)-norm, adversarial examples could be very similar to the corresponding correctly predicted inputs.

3.4.3 Carlini–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:

\[
\text{minimize } (1 - \beta) \cdot L_2(x', x)^2 + \beta \cdot f(y^*, M(x')) \\
\text{such that } x' \in [0, 1]^n
\]

where \( f \) is defined as:

\[
f(x') = \max(\max\{Z(x'_i) : \forall i \neq y^*\} - Z(x')_{y^*}, -t)
\]

where \( \beta \) is the weight to balance two terms, \( t \) is used to control the confidence of the output, \( Z(.) \) returns the pre-softmax value of the output layer, term \( \max\{Z(x'_i) : \forall i \neq y^*\} \) is the largest pre-softmax value except the target neuron, and term \( Z(x')_{y^*} \) is the pre-softmax value of target neuron.

3.4.4 Adversarial transformation network

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

\[
\sum_x \beta \cdot L_2(x, x') + L_2(M(x), r_\alpha(M(x), y^*))
\]

where \( \beta \) is a weight to balance two terms and function \( r_\alpha(.) \) modifies \( M(x) \) with the expectation that the modification on the correctly predicted input \( x \) is the least:

\[
r_\alpha(M(x), y^*) = \alpha \cdot \max\{M(x)\} \cdot \text{if } i = y^* \text{ otherwise } M(x)_{i \in [0, k-1]}
\]

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

However, this method does not support modifying specific portions of a correctly predicted input to generate an adversarial example. Instead, this method adds perturbation to all pixels of the correctly predicted input. Adding perturbation
to all pixels could be considered a pattern. This paper generalizes ATN to support different patterns, in which a pattern defines a set of modified pixels.

4 Pattern-based adversarial transformation network

To generate high-quality and diverse adversarial examples, PatternAttack consists of two proposals named the generalized ATN and the adversary optimizer. The illustration of PatternAttack is shown in Fig. 4. In the first phase, the generalized ATN 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 redundant adversarial pixels. Therefore, in the second phase, the proposed optimizer is employed to improve the quality of these adversarial examples. The proposed optimizer identifies redundant adversarial pixels and restores them to their original values. By removing these pixels, the resulting adversarial examples are still classified as the target label. Additionally, they are more similar to the correctly predicted inputs than the prior adversarial examples.

4.1 Generalized ATN

In the first phase of PatternAttack, the generalized ATN is applied to produce a set of adversarial examples. The architecture of the generalized ATN is based on a stacked convolutional autoencoder. These adversarial examples must satisfy a specific pattern. The objective of the generalized ATN satisfies two requirements. The first requirement is that the adversarial examples should be similar to the correctly predicted inputs 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 the generalized ATN is defined as follows:

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

(12)

where $\mathcal{P}$ is the pattern, CE is cross-entropy, and $\text{res}$ is the reconstructed image and computed by $ae(\gamma(\mathcal{P}, x))$ ($ae$ is the training autoencoder).

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

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

(13)

The adversarial example $x'$ is computed by using the equation $x - \gamma(\mathcal{P}, x) + \gamma(\mathcal{P}, \text{res})$. This equation replaces the pixels on $x$ satisfying the pattern $\mathcal{P}$ with new values returning from the autoencoder $ae$. All pixels not satisfied by the pattern $\mathcal{P}$ remain unchanged. This research uses three patterns including all-pixel pattern, object boundary pattern, and class model map pattern:

- **All-pixel pattern**: all pixels could be modified to generate adversarial examples. ATN uses this pattern to generate adversarial examples.
Object boundary pattern: all pixels located at the edge of objects on correctly predicted inputs would be modified. For example, on the MNIST dataset (Lecun et al. 1998b), objects are digits.

Class model map pattern (Simonyan et al. 2013): Let $I$ be a class model map of correctly predicted inputs. These correctly predicted inputs are classified as label $y_{true}$. The dimension of $I$ is the same as the dimension of the correctly predicted inputs. The class model map $I$ is generated by minimizing the following objective function:

$$\text{argmax}_I S_{y_{true}}(I) - \lambda \cdot L_2(I)^2$$

(14)

where $S_{y_{true}}$ is the score of the label $y_{true}$, $\lambda$ is a hyper-parameter, and $I$ is initialized as a zero image. The class model map $I$ is constructed by minimizing Eq. 14. The authors use back-propagation to find out $I$.

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

Compared to Eqs. 10, 12 has the following differences:

- The weight between terms: In our objective function, the weights are $(1 - \epsilon)$ for the first term and $\epsilon$ for the second term. In ATN, the first term has the weight $\beta$ and the second term has the weight 1. Essentially, our 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.
- Cross-entropy: In the second term of Eq. 10, while ATN uses $L_2$-norm, PatternAttack uses cross-entropy function. This would lead to better optimization of Eq. 12.
- Pattern: In Eq. 10, all pixels could be modified. Otherwise, in Eq. 12, only pixels satisfying a pattern $P$ would be modified.
- Function $r_\alpha$: This function is used in Eq. 10. Equation 12 does not use function $r_\alpha$. We only consider this function as an optional choice. We find out that using $r_\alpha$ does not produce a better result than not using $r_\alpha$ when attacking CNN models based on patterns. Instead of using the function $r_\alpha$, this paper uses a general strategy by proposing an optimizer, which improves the quality of adversarial examples in terms of $L_0$-norm and $L_2$-norm.

### 4.2 Adversarial example improvement

The adversarial examples generated by the generalized ATN might contain redundant perturbation. Hence, this section presents the proposed optimizer 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 redundant adversarial pixels are restored to the original values, the resulting adversarial examples are still classified as the target label $y^*$. In other words, these redundant adversarial pixels do not contribute to or only have 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 correctly predicted inputs as much as possible. Therefore, in order to improve the quality of adversarial examples, the pixels containing redundant perturbation should be restored to their original values.

#### Algorithm 1

**Input:** correctly predicted input $x$, adversarial example $x'$, target label $y^*$, CNN $M$, step $\alpha$, threshold $\delta$, and decay rate $t$

**Output:** An improved adversarial example

1. while $\alpha > \delta$ do
2. $S \leftarrow \text{COMPUTE_DIFF_FEATURES}(x, x')$
3. $S' \leftarrow \text{RANK_FEATURES}(S)$
4. $\# \text{block} \leftarrow |S'| / \alpha$
5. for $i = 0, i < \# \text{block}, i \leftarrow i + 1$ do
6. $\text{start} \leftarrow \alpha \cdot i$
7. $\text{end} \leftarrow \text{start} + \alpha$
8. if $\text{end} > |S'|$ then
9. $\text{end} \leftarrow |S'|$
10. end if
11. $x_{\text{clone}}' \leftarrow x'$
12. $x_{\text{start...end-1}}' \leftarrow x_{\text{start...end-1}}$
13. if $\text{argmax}_M(x') \neq y^*$ then
14. $x' \leftarrow x_{\text{clone}}'$
15. end if
16. if $\text{is_converged}$ then
17. return $x'$
18. end if
19. end for
20. $\alpha \leftarrow \alpha / t$
21. end while
22. return $x'$
This research proposes Algorithm 1 to improve the quality of adversarial examples in terms of $L_0$-norm and $L_2$-norm. The inputs of this algorithm are an adversarial example $\mathbf{x}'$, the correctly predicted input $\mathbf{x}$ of $\mathbf{x}'$, a target label $y^*$, an attacked model $\mathbf{M}$, a step size $\alpha \geq 1$, a positive threshold $\delta < \alpha$, and a decay rate $t > 1$. The output of the algorithm is an improved adversarial example. If $\alpha$ is greater than $\delta$ and the restored rate is not converged, the algorithm detects adversarial pixels by comparing $\mathbf{x}$ and $\mathbf{x}'$, and then stores these pixels in a set $S$ (line 2). After that, the adversarial pixels of $S$ are ranked by applying adversarial pixel 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 restores sequentially adversarial pixels in $S'$. A set of these sequentially adversarial pixels is called a block. For each block starting from position start to position end, in which the size of block is step $\alpha$ (line 7–11), the algorithm updates the adversarial example $\mathbf{x}'$ by restoring the block (line 12). If the restoration changes the classification, the current adversarial pixels are reverted to their original values (line 13–15). Next, $\alpha$ is decreased by $t$ times (line 20).

Algorithm 1 terminates when $\alpha$ is not greater than threshold $\delta$ (line 1) or the restored rate (denoted by $r$) converges (line 16). At the $i$-th modification, the restored rate $r$ of an adversarial example improvement is defined as follows:

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

(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:

$$\bigwedge (r(\mathbf{x}, \mathbf{x}', i) - r(\mathbf{x}, \mathbf{x}', i - 1) \leq \beta) \mid i \in [p - k + 1, p]$$

(16)

where $k \geq 1$ is number of prediction with the improvement of adversarial examples less than $\beta$.

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

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

For the pixel $\mathbf{x}_i$ on a correctly predicted input, there are several well-known pixel 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 pixels on the correctly predicted input. Specifically, for the pixel $\mathbf{x}_i$ which should increase to generate an adversarial example, JSMA$^+$ assigns this pixel to a non-zero score. By contrast, for the pixel $\mathbf{x}_i$ which should decrease to generate an adversarial example, JSMA$^-$ assigns this pixel to a non-zero score. Concerning COI, the score of a pixel $\mathbf{x}_i$ is computed by the multiplication of its intensity and its gradient.

However, the above heuristics do not consider ranking adversarial pixels. In this research, we apply these heuristics in a different context. We would compute the scores of adversarial pixels rather than the scores of pixels on the correctly predicted inputs. The score of an adversarial pixel $\mathbf{x}'_i$ is defined as follows:

- **JSMA$^+$**:
  $$s_y^*(\mathbf{x}'_i) = \begin{cases} 0 & \text{if } \frac{\partial M_y^*(\mathbf{x}')}{\partial \mathbf{x}'_i} < 0 \text{ or } \sum_{j \neq y^*} \frac{\partial M_j^*(\mathbf{x}')}{\partial \mathbf{x}'_i} > 0 \\ \frac{\partial M_y^*(\mathbf{x}')}{\partial \mathbf{x}'_i} \mid \sum_{j \neq y^*} \frac{\partial M_j^*(\mathbf{x}')}{\partial \mathbf{x}'_i} & \text{otherwise} \end{cases}$$
  (17)

- **JSMA$^-$**:
  $$s_y^*(\mathbf{x}'_i) = \begin{cases} 0 & \text{if } \frac{\partial M_y^*(\mathbf{x}')}{\partial \mathbf{x}'_i} > 0 \text{ or } \sum_{j \neq y^*} \frac{\partial M_j^*(\mathbf{x}')}{\partial \mathbf{x}'_i} < 0 \\ \frac{\partial M_y^*(\mathbf{x}')}{\partial \mathbf{x}'_i} \mid \left( \sum_{j \neq y^*} \frac{\partial M_j^*(\mathbf{x}')}{\partial \mathbf{x}'_i} \right) & \text{otherwise} \end{cases}$$
  (18)

- **COI**:
  $$s_y^*(\mathbf{x}'_i) = x'_i \cdot \frac{\partial M_y^*(\mathbf{x}')}{\partial \mathbf{x}'_i}$$
  (19)

5 Experiments

The experiments compare PatternAttack with other state-of-the-art methods including targeted FGSM (Goodfellow et al. 2015), box-constrained L-BFGS (Szegedy et al. 2014), and Carnili-Wagner $L_2$ (Carlini and Wagner 2016). 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 PatternAttack, the experiment addresses the following questions:

- **Diversity Improvement**: Can the generalized ATN generate diverse adversarial examples in comparison with targeted FGSM, box-constrained L-BFGS, and Carnili-Wagner $L_2$? (denoted by RQ1)
- **Quality Improvement**: Does the proposed optimizer improve the quality of adversarial examples effectively? (denoted by RQ2)
- **Overall Performance Analysis**: Does the proposed PatternAttack achieve good performance compared to targeted FGSM, box-constrained L-BFGS, and Carnili-Wagner $L_2$? (denoted by RQ3)

### 5.1 Dataset and attacked models

#### 5.1.1 Dataset

The experiments are conducted on the MNIST dataset (Lecun et al. 1998b) and CIFAR-10 dataset (Krizhevsky et al. 2009) to demonstrate the advantages of PatternAttack. These two datasets are commonly used 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 dimension (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 dimension (32, 32, 3) on the test set. Each pixel is represented in a combination of three colours including red, green, and blue. There are 10 labels including aeroplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

#### 5.1.2 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 2. As can be seen, on the MNIST dataset, LeNet-5 produces better results than AlexNet. On the CIFAR-10 dataset, AlexNet outperforms LeNet-5.

The result of attacks $m \rightarrow n$ is stacked convolutional autoencoders. These autoencoders reconstruct correctly predicted inputs 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 $M_i$ be a subset of the MNIST dataset and $C_i$ 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 $G_{train}$, $G_{val}$, and $G_{new}$. Group $G_{train} = \{M_{train}, C_{train}\}$ is a portion of the training sets and used to train autoencoders. Each dataset of $G_{train}$ consists of 1,000 correctly predicted inputs labelled as $m$. Group $G_{val} = \{M_{val}, C_{val}\}$ is a portion of the training sets and not used to train autoencoders. Each dataset of $G_{val}$ contains 4,000 correctly predicted inputs labelled as $m$. Group $G_{new} = \{M_{new}, C_{new}\}$ is the test sets of the attacked models. $M_{new}$ has 1,000 correctly predicted inputs labelled as $m$. Due to the accuracy limitation of two CNNs on CIFAR-10 test set, we set $C_{new}$ to have 862 and 759 correctly predicted inputs labelled as $m$ for AlexNet and LeNet-5, respectively. Group $G_{val}$ and Group $G_{new}$ are used to evaluate the generalization ability of the autoencoders trained on Group $G_{train}$.

For each CNN model, there are 90 attacks in total (i.e., 10 original labels × 9 target labels). However, in this research, we select 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. Because the shape of 9 is very different from 7 and 4, it is likely hard to perform adversarial attacks on 9 digits. On the CIFAR-10 dataset, the attack is truck $\rightarrow$ horse for AlexNet and truck $\rightarrow$ deer for LeNet-5. Because the shape of the truck is very different from the horse and the deer, it is likely hard to perform attacks on truck images.

### 5.2 Configuration of PatternAttack

#### 5.2.1 Generalized ATN

**Architecture of generalized ATN**: It is difficult to select an architecture of a stacked convolutional autoencoder which

| 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%   |
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 Tables 3 and 4. Group $G_{train}$ is used to train autoencoders. Each autoencoder is trained up to 500 epochs with a batch size of 256. The experiment uses an early stopping strategy to terminate the training when there is no improvement in the loss over subsequent epochs. After the training process, correctly predicted inputs 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.

The value of weight $\epsilon$: We need to fine-tune weight $\epsilon$ in Eq. 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 $\epsilon$ equals 0.01 or 0.05, the attacks tend to minimize the distance term. If $\epsilon$ equals 0.5, the attacks minimize the two terms with equal weights. If $\epsilon$ equals 0.95 or 0.99, the attacks tend to minimize the cross-entropy term. If $\epsilon$ equals 1, the attacks only minimize the cross-entropy term.

**Pattern:** The experiments use three different patterns to generate adversarial examples including all-pixel pattern, object boundary pattern, and class model map pattern. For class model map pattern, this pattern describes the impact of pixels on the classification of the attacked models. On the MNIST dataset, we empirically choose the top-100 pixels on the class model map. The attacks would add perturbations to 100 fixed pixels to generate adversarial examples. On the CIFAR-10 dataset, we empirically choose the top-400 pixels on the class model map because the number of pixels on this dataset is about four times larger than that of the MNIST dataset.

### 5.2.2 Adversary improvement

The proposed optimizer is used to improve the quality of adversarial examples in terms of $L_0$-norm and $L_2$-norm. The proposed optimizer uses four main parameters including step $\alpha$, threshold $\delta$, decay rate $\tau$, and ranking heuristics. These parameters are used to adjust the performance of the proposed optimizer. In our experiment, the value of $\delta$ is assigned to 0. The value of decay rate $\tau$ is assigned to 2.

**Step $\alpha$:** By analyzing the result of different steps $\alpha$, we observed that using the appropriate value of step $\alpha$ would have a positive impact on the performance of the proposed optimizer. Using a too-large value of step $\alpha$ or a too-small value of step $\alpha$ might slow down the performance of the proposed optimizer. 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 pixels in increasing order of impact on the target neuron, heuristics random sorts adversarial pixels 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 improves adversarial examples most effectively.

### 5.3 Configuration of compared methods

We compare PatternAttack with other state-of-the-art methods including targeted FGSM, box-constrained L-BFGS, and
5.4 Answers to research questions

5.4.1 RQ1: Improvement in diversity issue

This section investigates the ability of the generalized ATN to improve the low diversity of adversarial examples. In order to generate diverse adversarial examples, the experiment applies different patterns including all-pixel pattern, object boundary pattern, and class model map pattern. Table 5 compares the average success rates of the generalized ATN and other comparable methods. There are two promising results.

Firstly, the generalized ATN could generate adversarial examples with various perturbations than other methods. Specifically, all-pixel pattern achieves 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 average success rate of object boundary pattern achieves from 6.2% to 55.1%. Class model map pattern achieves the average success rate from 9.4% to 68.3%. If the generalized ATN only uses all-pixel pattern like Carnili-Wagner $L_2$, targeted FGSM, and L-BFGS, a large number of adversarial examples could be ignored.

Secondly, the generalized ATN could generate adversarial examples from new datasets with an average success rate approximate to the trained datasets. The trained datasets are in Group $G_{\text{train}}$ (i.e., $M_{\text{train}}$ and $C_{\text{train}}$). The new datasets are in Group $G_{\text{val}}$ (i.e., $M_{\text{val}}$ and $C_{\text{val}}$) and Group $G_{\text{new}}$ (i.e., $M_{\text{new}}$ and $M_{\text{new}}$). 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 the generalized ATN, which is not supported in comparable methods.

5.4.2 RQ2: Improvement in quality issue

This section investigates the ability of the proposed optimizer to enhance the quality of adversarial examples. The conclusion is that the proposed optimizer could enhance the quality of adversarial examples significantly for the generalized ATN, targeted FGSM, and box-constrained L-BFGS. For the
Table 6 The average reduction rate (%) of adversarial examples by applying the proposed optimizer. Larger values are better

| Model                  | Method                                         | $L_0$ | $L_2$ |
|------------------------|------------------------------------------------|-------|-------|
| AlexNet $9 \rightarrow 7$ | Generalized ATN+ object boundary | 75.9  | 42.9  |
|                        | Generalized ATN+ class model map              | 84.4  | 51.0  |
|                        | Generalized ATN+ all-pixel (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 $9 \rightarrow 4$ | Generalized ATN+ object boundary              | 70.1  | 38.9  |
|                        | Generalized ATN+ class model map              | 71.9  | 36.9  |
|                        | Generalized ATN+ all-pixel (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 $\rightarrow$ horse | Generalized ATN+ object boundary | 96.3  | 78.5  |
|                        | Generalized ATN+ class model map              | 85.8  | 60.2  |
|                        | Generalized ATN+ all-pixel (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 $\rightarrow$ deer | Generalized ATN+ object boundary | 95.7  | 78.0  |
|                        | Generalized ATN+ class model map              | 88.4  | 62.4  |
|                        | Generalized ATN+ all-pixel (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   |

adversarial examples generated by Carnili-Wagner $L_2$, the proposed optimizer could enhance their quality slightly.

In the proposed optimizer, rather than improving the quality of adversarial examples individually, we put them in a batch and improve them 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 6 shows the average reduction rate of $L_0$-norm and $L_2$-norm when applying the proposed optimizer. Generally, $L_0$-norm and $L_2$-norm could be reduced by applying the proposed optimizer. Especially, concerning $L_0$-norm, we observed that the proposed optimizer could decrease from a large number of adversarial pixels to one adversarial pixel. For the generalized ATN, targeted FGSM, and box-constrained L-BFGS, the proposed optimizer could enhance the quality of the generated adversarial examples significantly. The proposed optimizer could decrease at most about 98% $L_0$-norm and 89% $L_2$-norm. For Carnili-Wagner $L_2$, the proposed optimizer 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 Eq. 8, which leads to a minimum $L_2$-norm.

Examples of some improvements for adversarial examples are shown in Fig. 6. Labels origin and adversary denote the correctly predicted inputs and their corresponding adversarial examples, respectively. Label difference shows the difference between adversarial examples and their improvement in white colour.

5.4.3 RQ3: Overall performance analysis

Table 7 shows the overall performance of PatternAttack and other comparable methods. Generally, PatternAttack could achieve good average success rates while requiring lower computational cost considerably compared to the other methods. This is a promising result to show that PatternAttack could be applied in practice.

We need to evaluate the overall performance of PatternAttack when dealing with new datasets. For all patterns, the experiment shows that PatternAttack could generate adversarial examples from new datasets (i.e., Group $G_{val}$...
Improving diversity and quality of adversarial examples generated by the proposed optimizer

Fig. 6 Examples of improved adversarial examples generated by the proposed optimizer

Table 7 The overall performance comparison (seconds) between PatternAttack and other methods

| Model        | Dataset | PatternAttack | Targeted FGSM | L-BFGS |
|--------------|---------|---------------|---------------|--------|
|              |         | Object boundary pattern | Class model pattern | All-pixel pattern (ATN) | Camili–Wagner $L_2$ | Targeted FGSM | L-BFGS |
| AlexNet 9 → 7 | $M_{train}$  | 98            | 131           | 174    | 1,815 | 18  | 240   |
|              | $M_{val}$   | 21            | 38            | 87     | 7,176 | 61  | 1,009 |
|              | $M_{new}$   | 11            | 21            | 63     | 1,716 | 20  | 258   |
| LeNet-5 9 → 4 | $M_{train}$  | 84            | 109           | 95     | 714   | 11  | 141   |
|              | $M_{val}$   | 13            | 9             | 55     | 2,597 | 40  | 549   |
|              | $M_{new}$   | 6             | 4             | 46     | 728   | 12  | 128   |
| AlexNet truck → horse | $C_{train}$  | 216           | 287           | 309    | 2,255 | 18  | 289   |
|              | $C_{val}$    | 26            | 55            | 125    | 8,942 | 68  | 1,152 |
|              | $C_{new}$    | 13            | 24            | 99     | 2,164 | 17  | 324   |
| LeNet-5 truck → deer | $C_{train}$  | 173           | 181           | 190    | 917   | 42  | 158   |
|              | $C_{val}$    | 13            | 44            | 53     | 3,276 | 195 | 622   |
|              | $C_{new}$    | 8             | 21            | 47     | 919   | 40  | 148   |
and Group \( G_{\text{new}} \) with lower computational cost than the trained datasets (i.e., Group \( G_{\text{train}} \)). The computational cost on Group \( G_{\text{train}} \) includes the training process, the adversarial example generation process, and the improvement process for adversarial examples. The computation cost on Group \( G_{\text{val}} \) and Group \( G_{\text{new}} \) 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 \( G_{\text{val}} \) and Group \( G_{\text{new}} \) is from 2 times to 27 times faster than that of Group \( G_{\text{train}} \). 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 correctly predicted inputs with an extremely low computational cost. By contrast, the other methods do not support generating adversarial examples based on the information from previous adversarial example generation. In other words, every adversarial example is generated independently of the other. As a result, when dealing with a set of correctly predicted inputs, PatternAttack usually produces adversarial examples faster than the other methods.

Another promising finding is that PatternAttack 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 PatternAttack 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 5, 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 applying all-pixel pattern, PatternAttack 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-pixel pattern is from 7.5 times (i.e., LeNet-5 with dataset \( M_{\text{train}} \)) to 80 times (i.e., AlexNet with dataset \( M_{\text{val}} \)) faster than Carnili-Wagner \( L_2 \).

6 Discussion

Integral constraint: On the MNIST dataset and the CIFAR-10 dataset, the value of each pixel is an integer in the set \( \mathcal{D} = \{0, 1, ..., 255\} \). After normalization, the value of each pixel is in the set \( \mathcal{D}' = \{0, \frac{1}{255}, \frac{2}{255}, ..., 1\} \). The value of an adversarial pixel must be in the valid set \( \mathcal{D} \) or \( \mathcal{D}' \). However, similarly 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 \( \mathbf{M} \left( \frac{\mathbf{x}' + 255}{255} \right) = y^* \), 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 correctly predicted inputs 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 PatternAttack method to generate high-quality and diverse adversarial examples. PatternAttack has two proposals including the generalized ATN and the adversary optimizer. Firstly, the generalized ATN is a pattern-based stacked convolutional autoencoder. This autoencoder is used to generate diverse adversarial examples for CNN. We conducted experiments with three patterns including all-pixel pattern, object boundary pattern, and class model map pattern. Secondly, we introduce the optimizer to enhance the quality of adversarial examples. The main idea of the proposed optimizer is to restore redundant adversarial pixels 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, the generalized ATN could generate diverse adversarial examples by using different patterns. Especially, for all-pixel pattern, most of the attacks could achieve above 99% average success rate. In terms of adversarial quality, concerning \( L_2 \)-norm, the proposed optimizer could reduce from hundreds of adversarial pixels to one adversarial pixel. About \( L_2 \)-norm, the proposed optimizer could reduce the average distance considerably. The proposed optimizer 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 pixel ranking heuristics COI and JSMA in the proposed optimizer to achieve better performance than using the random heuristics. In the future, PatternAttack will be extended to support other distance metrics such as \( L_{\infty} \)-norm. In addition, the research will make a more comprehensive comparison with other well-known datasets.
Acknowledgements This work is supported by Ministry of Science and Technology, Vietnam under project number KC-4.0-07/19-25. Program KC4.0/19-25. 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.2022.TS001. 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.

Author Contributions Conceptualization: PNH, D-AN; Methodology: D-AN, KDM; Formal analysis and investigation: D-AN, KDM; Writing - original draft preparation: D-AN; Writing - review and editing: all authors.

Funding 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.2022.TS001. 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.

Data Availability All data and material are available.

Code Availability All source code are available.

Declarations

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

Ethics approval Not applicable.

Informed consent Not applicable

References

Akhatar N, Mian A, Kardan N, Shah M (2021) Threat of adversarial attacks on deep learning in computer vision: Survey ii.
Aldahdooh A, Hamidouche W, Fezza S A, Déforges O (2021) Adversarial example detection for DNN models: a review. CoRR arXiv:2105.00203
Andrew Ng. (2011) Sparse autoencoder. CS294A Lecture notes
Baluja S, Fischer I (2017) Adversarial transformation networks: learning to generate adversarial examples
Bengio Y, Lamblin P, Popovici D, Larochelle H (2006) Greedy layerwise training of deep networks. In: Proceedings of the 19th international conference on neural information processing systems, NIPS’06, page 153–160, Cambridge, MA, USA, MIT Press
Cao C, Liu X, Yang Y, Yu Y, Wang J, Wang Z, Huang Y, Wang L, Huang C, Xu W, Ramanan D, Huang T S (2015) Look and think twice: capturing top-down visual attention with feedback convolutional neural networks. In: ICCV, pp. 2956–2964.
Carlini N, Wagner D A. (2016) Towards evaluating the robustness of neural networks. In: Proceedings of the 19th international conference on neural information processing systems, NIPS’06, page 153–160, Cambridge, MA, USA, MIT Press
Cao C, Liu X, Yang Y, Yu Y, Wang J, Wang Z, Huang Y, Wang L, Huang C, Xu W, Ramanan D, Huang T S (2015) Look and think twice: capturing top-down visual attention with feedback convolutional neural networks. In: ICCV, pp. 2956–2964.
Carlini N, Wagner D A. (2016) Towards evaluating the robustness of neural networks. In: Proceedings of the 19th international conference on neural information processing systems, NIPS’06, page 153–160, Cambridge, MA, USA, MIT Press
Dabkowski P, Gal Y (2017) Real time image saliency for black box classifiers
Dong Y, Liao F, Pang T, Hu X, Zhu J (2017) Discovering adversarial examples with momentum. CoRR arXiv:1710.06081
Ettraneck C, Lenz S, Maass P, Schönlieb C-B (2019) On the connection between adversarial robustness and saliency map interpretability
Fong R, Vedaldi A (2017). Interpretable explanations of black boxes by meaningful perturbation. CoRR arXiv:1704.03296
Goodfellow Ian J., Shlens Jonathon, Szegedy Christian (2015) Explaining and harnessing adversarial examples
Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, London
Gopinath Divya, P˘as˘areanu Corina S., Wang Kaiyuan, Zhang Mengshi, Khurshid Sarfraz (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, pp. 282–283. IEEE Press, https://doi.org/10.1109/ICSE-Companion.2019.00115
Gu Jindong, Tresp V (2019) Saliency methods for explaining adversarial attacks. CoRR arXiv:1908.08413
IEEE. (1990) Ieee standard glossary of software engineering terminology.
Kingma D P, Welling M (2014) Auto-encoding variational bayes
Krizhevsky A, Sutskever I, Hinton GE (2017) Imagenet classification with deep convolutional neural networks. Commun ACM 60(6):84–90. https://doi.org/10.1145/3065386
Krizhevsky A, Nair V, Hinton G (2009) Cifar-10. Canadian Institute for Advanced Research
Kurakin A, Goodfellow I J., Bengio S, (2016) Adversarial examples in the physical world. CoRR arXiv:1607.02533
Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324. https://doi.org/10.1109/5.726791
Lecun Y, Bottou Léon, Bengio Y, Haffner P (1998b) Gradient-based learning applied to document recognition. In: Proceedings of the IEEE, pp. 2227–2324
Li X, Ji S (2020) Defense-vae: a fast and accurate defense against adversarial attacks. In: Peggy C, Kurt D (eds) Machine learning and knowledge discovery in databases. Springer, Cham, pp 191–207
Masci J, Meier U, Ciresan D, Schmidhuber J (2011) Stacked convolutional auto-encoders for hierarchical feature extraction. In Timo H, Włodzisław D, Mark G, and Samuel K, (eds), Artificial neural networks and machine learning – ICANN 2011, pp.52–59, Berlin, Heidelberg, Springer Berlin Heidelberg
Moosavi-Dezfooli S-M, Fawzi A, Frossard P (2015) Deepfool: a simple and accurate method to fool deep neural networks. CoRR arXiv:1511.04599
Moura L, Bjoørner N (2008) Z3: an efficient smt solver. In: Proceedings of the 19th international conference on automated software engineering, pages 37–46, Washington, DC, USA, IEEE Computer Society
Nogueira F, Sultana F, Sufian A, Dutta P (2019) Symbolic execution-based method to perform untargeted attack on feedforward neural networks. Autom Software Eng, 29
Papernot N, McDaniel P D., Jha S, Fredrikson M, Celik Z. B, Swami A (2015) The limitations of deep learning in adversarial settings. CoRR arXiv:1511.07528
Pei K, Cao Y, Yang J, Jana S (2017) Deepxplore: automated whitebox testing of deep learning systems. CoRR arXiv:1705.06640
Pu Y, Wang W, Henaø R, Chen L, Gan Z, Li C, Carin L (2017) Adversarial symmetric variational autoencoder. CoRR arXiv:1711.04915
Simonyan K, Vedaldi A, Zisserman A (2013) Deep inside convolutional networks: visualising image classification models and saliency maps. CoRR arXiv:1312.6034
Springenberg J T, Dosovitskiy A, Brox T, Riedmiller M (2014) Striving for simplicity: the all convolutional net. arXiv preprint arXiv:1412.6806
Sultana F, Sufian A, Dutta P (2019) Advancements in image classification using convolutional neural network. CoRR arXiv:1905.03288
Su J, Vargas D V, Sakurai K (2017) One pixel attack for fooling deep neural networks. CoRR arXiv:1710.08864
Szegedy C, Zaremba W, Sutskever I, Bruna J, Goodfellow I, Rob F (2014) Intriguing properties of neural networks, Dumitru Erhan Tolstikhin I, Bousquet O, Gelly S, Schoelkopf B (2019) Wasserstein auto-encoders
Tsipras D, Santurkar S, Engstrom L, Turner A, Madry A (2019) Robustness may be at odds with accuracy
Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol P-A (2010) Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J Mach Learn Res 11:3371–3408
Yu F, Dong Q, Chen X (2018) ASP: a fast adversarial attack example generation framework based on adversarial saliency prediction. CoRR arXiv:1802.05763
Zeiler M D, Fergus R (2013) Visualizing and understanding convolutional networks
Zhang J, Harman M, Ma L, Liu Y (2019) Machine learning testing: survey, landscapes and horizons, 06
Zhang J, Lin Z, Brandt J, Shen X, Sclaroff S (2016) Top-down neural attention by excitation backprop. CoRR arXiv:1608.00507

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.