THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Practical Adversarial Attacks Generation to Machine Learning Models

by

Tao Cheng

School of Information Technology and

Electrical Engineering

University of Queensland.

Submitted for the degree of

Master of Computer Science

7 November 2022

Tao Cheng

S4718266@student.uq.edu.au

7 November 2022

Prof Michael Bruenig

Head of School

School of Information Technology and Electrical Engineering

The University of Queensland

St. Lucia QLD 4072

Dear Professor Bruenig,

In accordance with the requirements of the Degree of Master of computer science in the School of Information Technology and Electrical Engineering, I submit the following thesis entitled

"Practical Adversarial Attacks Generation to Machine Learning Models"

The thesis was performed under the supervision of Associate Professor Dan Kim. I declare that the work submitted in the thesis is my own, except as acknowledged in the text and footnotes, and that it has not previously been submitted for a degree at the University of Queensland or any other institution.

Yours sincerely

Tao Cheng

Acknowledgements

I would like to thank Dr. Dan Kim for his help in finishing this project. His feedbacks and suggestions helped me with considering ideas to explore the algorithm and improving it. From little knowing what adversarial attack is, to learning, proposing, implementing, and evaluating an attack generation method with some improvements in only one-semester long. I learned much about not only adversarial attacks generation and implementation, but also realize the importance of machine learning models robustness. Finally, this great experience taught me the problem-solving ability, which will help me in the future career.

Abstract

With the development of machine learning, convolutional neural network (CNN) based methods have achieved great progress recently. However, this also leads to machine learning models being exposed to malicious attacks, especially adversarial attacks have become one of the most popular evasion attacks. In this paper, I explore the insight of the one-pixel attack, a black-box attack with a highly limited situation that only modifies one pixel of the original image by using a differential evolution algorithm. Compared to other popular white-box attacks such as Fast Gradient Sign Method (FGSM) and Basic Iterative Method (BIM), the original One-pixel black-box attack costs high computation and requires much time to achieve the goal. In this case, I try to improve the original pixel-attack method by changing the population of the candidate's selection for the target image and changing the decision criteria of the attack algorithm, which aims to make this attack more adaptive when facing real-world issues.

Contents

List of Figures

1 Introduction

Research and probe on adversarial examples for machine learning has been a popular topic in recent years. Preventing the attacks, especially the black-box attacks on machine learning models, can be a crucial part for AI models to approach Confidentiality, Integrity, and Availability. Previous works, such as Sun et al.'s article about practical adversarial example attacks [21], lacks checking the model's robustness to attacks generated. In this case, performing adversarial training with attacks generated can be evaluated with a new standard, namely the robustness, which can help us to understand the insight of the generation of adversarial examples and attacks. It is significant to build a reliable model/system to resist potential risks by developing decent adversarial attack method, which can train previous model to be more adaptive as malicious perturbations exist.

The adversarial attacks are generated by adding some noises or perturbations to benign images, which makes the raw image become an adversarial example. According to Goodfellow et al. [8], these adversarial examples can make the DNN become vulnerable, which causes damages to DNN misclassify. Some adversarial examples are easy to be perceptible to human eyes due to the pixel variation. However, some examples are difficult to detect by machine learning models and humans. In this case, adding only one tiny amount of well-designed perturbation for creating adversarial images has been proposed by Su et al. [20], namely the one-pixel attack.

In general, I proposed some improvements on the original one-pixel attack generation, which is changing the amount of the candidate selection of the attack algorithm and changing the decision criteria of the process of attacking the image. I tried to figure out whether these two methods can either increase the attack success rate or decrease the attack time, which are two main attributes to evaluate the performance of an adversarial attack.

2 Preliminaries

In this section, background and related works are introduced.

2.1 Machine Learning (ML)

According to Alpaydin, machine learning is the art of programs solving problems without being explicitly programmed, which is the development of computer systems that can learn and adapt by using algorithms and models to analyze the insight and inference from data's pattern [1]. In general, machine learning is used in a wide variety of applications nowadays, such as image recognition, speech recognition, self-driving automobiles, medicine diagnosis etc.

Among the course materials of machine learning, some popular models, such as linear regression, logistic regression, decision trees etc. However, these machine learning models can be vulnerable to inputs with purposely designed perturbations or noises and perform some unexpected outputs [1][5].

- **Deep learning (DL)**

Deep learning models are more capable of complex computing problems, compared with normal/traditional machine learning models. One significant difference between deep learning and machine learning is that deep learning models could complete features extraction by themselves, instead of manipulated that by developers. In this case, deep learning models have more capability than normal machine learning models in solving complex tasks. Although deep learning models' training will consume much more time by using larger datasets with massive trainable parameters, the output of deep learning models could perform much better than the traditional machine learning models' results.

However, both two models hold the same property, namely they can be fooled with noises and perturbations even they have great generalization performance [4].

- **Classification**

According to Alpaydin, the definition of classification problem is, a set of samples from different classes are used to train a model, which is expected to predict class labels for unseen data [1]. One popular classification problem example is customer behavior prediction, which is classifying different types of customers based on their buying items. In addition, classifiers are the main targets for most of the adversarial attacks, which will influence the result directly.

- **Regression**

Another big category of machine learning problem is the regression problem. Regression problem is, building a model to approximate the function, for prediction future values according to historical data or interpolate missing data [1]. One significant example in real-life is predicting the housing values. This type of models is vulnerable to adversarial attacks. According to Jagielski et al. [9], they proposed that poison attacks can have a massive performance on health care, loan & lease, and house pricing datasets.

- **Neural Networks**

One of the most popular deep learning models is neural networks model, which has been proved to show a great performance in the filed of solving machine learning problems.

The structure of neural networks is inspired by the biological neural networks, namely the human brains. Sets of neurons relate to weights in a neural network structure from nerves in the human brain. The output y for each neural unit corresponding to the input x, and the formula as follows:

$$y = w * x + b$$

The w represents weights and b represents biases. According to Senzaki et al. [17], these parameters are optimized through the learning process. Weights can control every input, for affecting the behavior of neural unit, namely neuron; bias is utilized to solve the scenario under which all inputs are zero-value, which can give a neuron more capable during the data processing.

The output of a neuron will pass to an activation function to determine whether it is on-active or "on fire". When an incoming signal meets the conditions, a neuron will send a message to the connected neurons. On the other hand, the signal is under the condition, no messages will be sent. In this case, the threshold of becoming active could be think of a decision boundary in classification problems, which will be divided into different classes of samples.

The figure (Figure 1) shown below represents a neural network consists of connected layers. An input vector passes through the input layer, hidden layer, and approaches the output layer. The final output of the network will be passed to an activation function to determine the result.

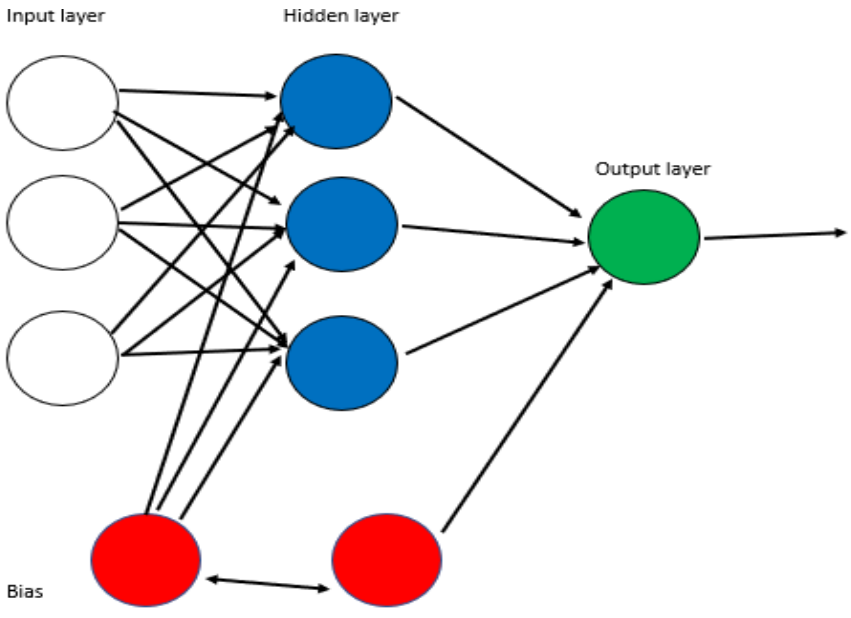


Figure 1. Structure of a simple neural network with bias nodes

There are many activation functions, which will reflect the optimization algorithm process through the training. Thus, the activation function's property will determine the performance and training speed of the neural network model.

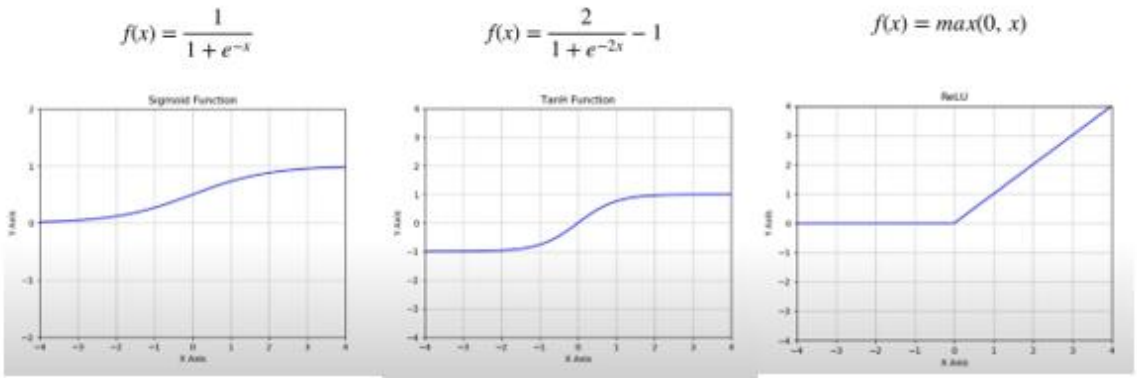


Figure 2. Sigmoid, TanH, and ReLU functions examples

Backpropagation can be the method of training the neural network. According to Bishop [3], the approach to the problem of determining the network parameters can be regarded as minimizing a sum-of-squares error function, which is one of the loss functions. When the data utilized by the model, a predefined loss function will compute the loss. After this step, the gradients will be calculated and used to update the weights for minimizing loss. Mean square error (MSE) loss function is the simplest one.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2. \qquad \text{MSE} = \frac{1}{n} * \sum_{k=0}^{n-1} (y^k - y^{k\prime})^2$$

Figure 3. Left: Sum-of-squares function; Right: Mean-square-error loss

The n is the number of samples, yk is the true output, and yk' is the output of the neural network model.

Learning rate controls the pace of each update during training. A model with a high learning rate may not be able to converge to the optimal answer. With a low learning

rate, training can take an unreasonable amount of time. Consequently, strategies such as momentum and adaptive learning rate for adjusting the learning rate are crucial [1].

- **Multilayer Perceptron**

A neural network which consists of fully connected layers only can be consider a multilayer perceptron. According to Ben-Yacoub et al. [2], normally MLPs' structure is not complex, but they have solid performance when solving detection assignments, such as face recognition and detection.

- **Convolutional Neural Networks**

Convolutional Neural Networks, as known as CNNs, are based on the regular neural networks with convolutional layers. In this case, they have a better performance than fully connected networks. Within the layers, each region of input with filters will be convolved by the scalar values, which result in optimized model through the learning process [15].

In machine learning, the concept of convolution is quite different from the concept in mathematics. For convolutional layers to function, kernels are essential. Kernels, in contrast to fully connected layers, where each node has a separate weight connecting it to the next layer, share inputs and have smaller weights. The stride, or step size, with which a kernel moves determines the form of the output vector. One example of the operation has shown below:

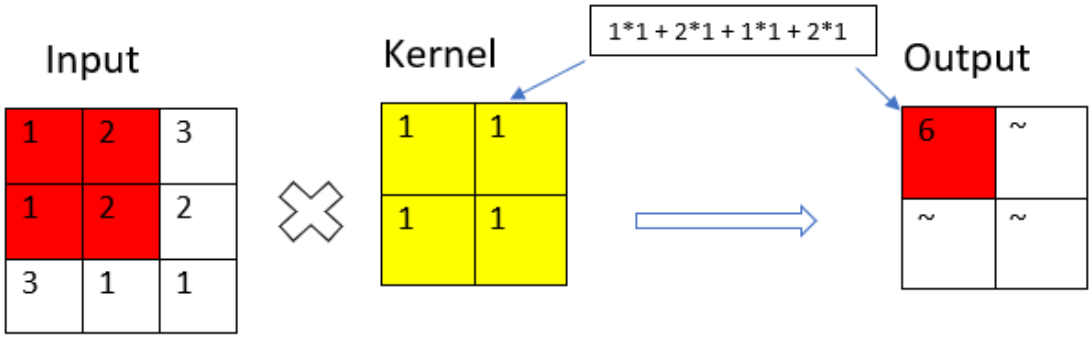


Figure 4. Example operation of Convolution with stride of 1

Form the Figure 4, the output vector becomes a lower dimension compared to the input vector. After the kernel shift, the final output will become 1-dimension, which only this 1-dimentional kernel can be implemented. The padding method is to solve the problem that adding edges with 0 value to the outside the input, which aims to adjust the final shape of the output.

These two methods, padding and stride, are used together to control the process of convolutional layers. Figure 5 and Figure 6 shows the example of same input size and kernel size, but with different output shapes.

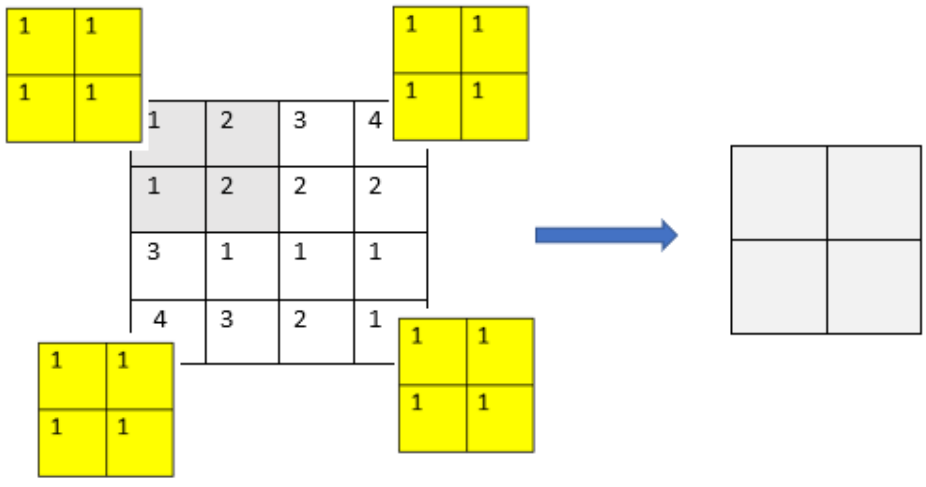


Figure 5. Example operation of Convolution with stride of 2

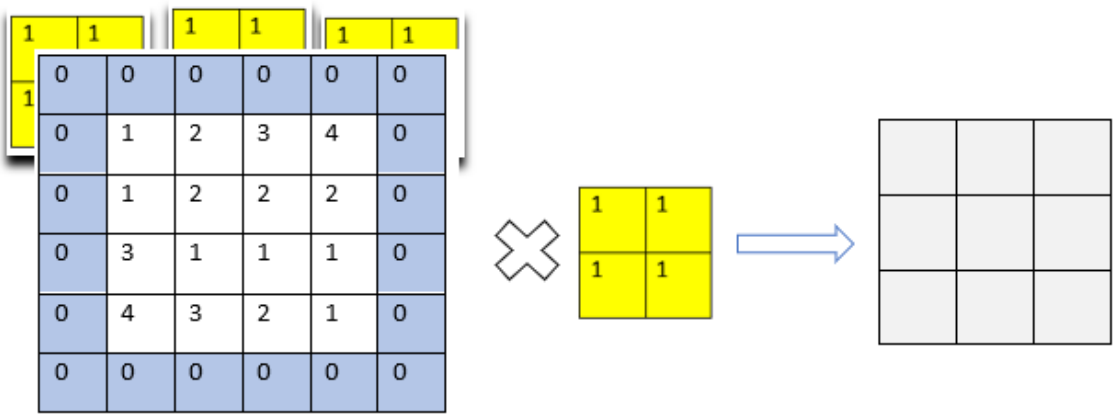


Figure 6. Example operation of Convolution with padding of 1 and stride of 2

It is claimed that the features extracted by convolutional layers assist the model get a deeper understanding of the data. These features are occasionally pooled to make them more representative. After a convolution layer, networks frequently employ a pooling procedure. By replacing many features with a single representative value, feature dimension is reduced; this is the core notion behind feature pooling.

Max pooling and average pooling are two effective and popular pooling method. Max pooling aims to output the maximal feature value from the kernel, while average pooling replaces the original features with their average feature value. Stride and padding still useful during the pooling operations, which restrict the shape of the convolution operation's output (Figure 7).

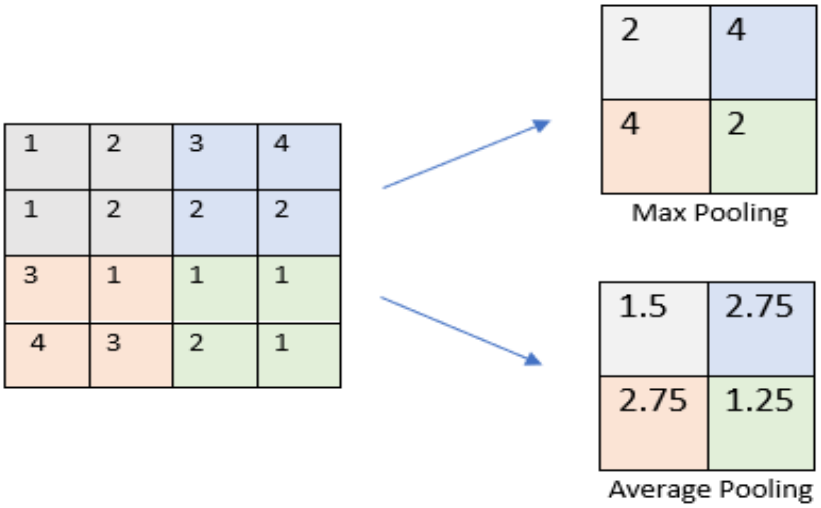


Figure 7. Example of max pooling and average pooling

Following the input layer, a convolutional network typically has numerous convolutional layers and several fully connected layers. Kernels extract significant features I the early section of a network; these features are then used for categorization by fully linked layers (Figure 8). However, convolutional networks are easy to be fooled by adversarial attacks, even they have great performance on solving complex problems.

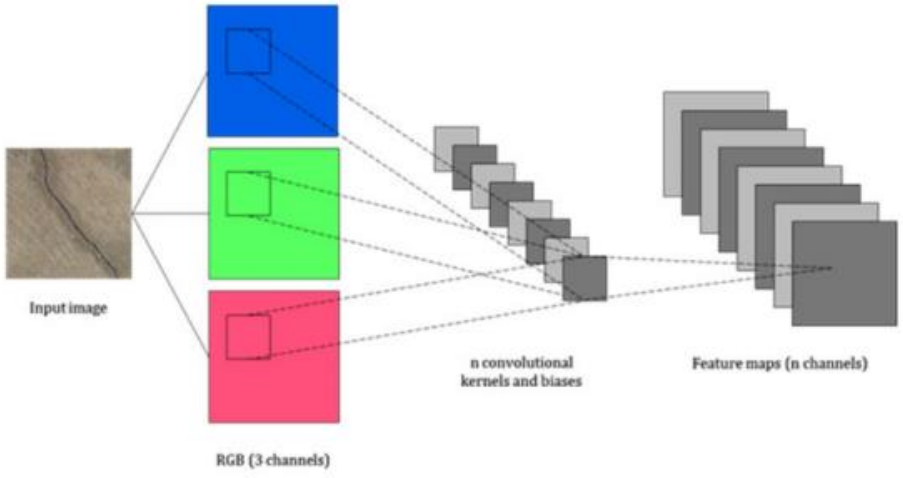


Figure 8. Example of Convolutional layer; Original image from: "Automatic Pixel-Level Crack Detection and Measurement Using Fully Convolutional Network: Pixel-level crack detection and measurement using FCN" [23].

2.2 Adversarial attacks in machine learning

- **Adversarial examples**

Normally, the adversarial examples are inputs to machine learning models that attackers make these models to cause mistakes with well-planned purpose. For example, a benign image with a perturbation or noise added to become a new image. And this image is the adversarial example (image) (see Figure 9).

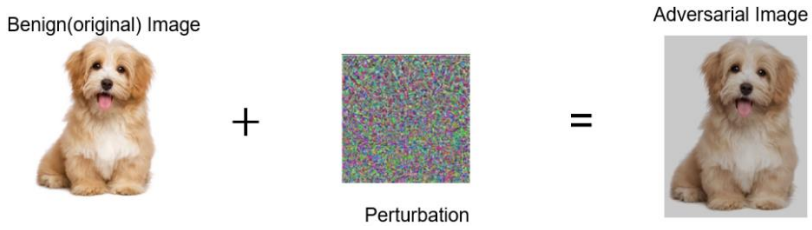


Figure 9. Adversarial example (images)

- **Adversarial attack**

The adversarial attack to machine learning models is the sample with designed perturbations, which mislead the classifier of models to a false output prediction. To be specific, a classifier can be treated as a function with training such as: y = f (x), which x is the input value and y is the output value. While an adversarial attack will present a perturbation sample x', which make the function becomes like:

$$f(x') = y',\ f(x) = y,\ \text{which } y' \neq y,$$

in the case of this, the true output y will be affected to y'.

- **White-box attack and Black-box attack**

Attackers acknowledge the information, such as gradients, of their target models during the attack generation, this action can be defined as white-box attacks. While attackers cannot access to the structure of models, only a few information is available of target models, will be classified as black-box attacks.

In real world, most adversarial attacks are black-box type, since it is quite impossible to access to running states of application or software, which is normally secured by anti-

attack software or firewalls. In this case, black-box attack generations are much difficult but more adaptive to face the real-world problems.

- **Untargeted attack and targeted attack**

The basic idea of untargeted attacks is that attackers only make the target models produce a false output or prediction. While the targeted attacks aim to make the targeted models output a specific result or prediction.

For example, a benign image X_0 was attacked, and the adversarial image, X_{adv} = X + noise, will be adopt to the model f. y is the true class of the image X_0, and y' is the result of the adversarial image X_{adv}. y'' is the targeted class (Figure 10).

For untargeted attack, the result will be like:

$$f(X_{adv}) = y', \text{ where } y' != y \text{ and } X_{adv} = X + noise$$

For targeted attack, the result will be like:

$$f(X_{adv}) = y', \text{ where } y' != y, y' == y'' \text{ and } X_{adv} = X + noise$$

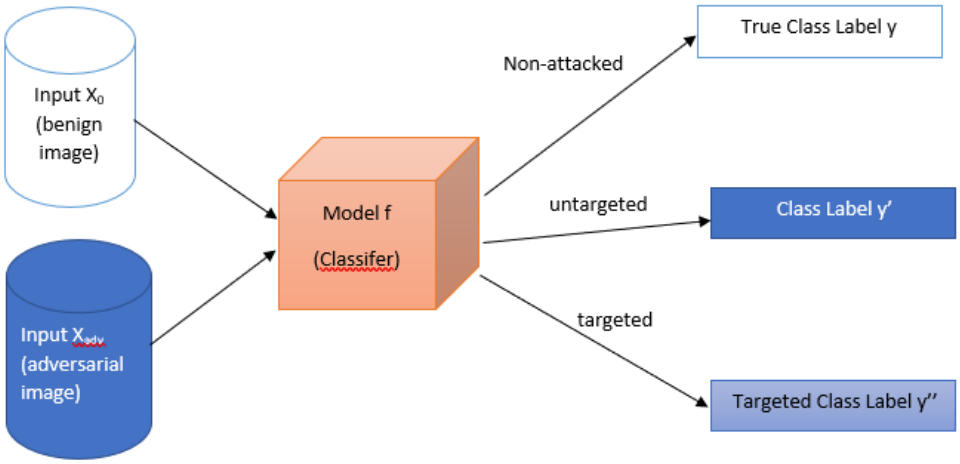


Figure 10. untargeted attack and targeted attack

- **Perturbations and Measurement**

Adversarial attacks with perturbations or noises generated can be measured by L_p norms, which is the metrics measuring magnitude of vectors in mathematics. L_0 norm is:

$$\text{L\_0} - \text{norm} = \sum_{i=1}^{d} |x_i|^0$$

which measures the number of modified pixels related to the adversarial image.

L_2 norm measures the average magnitude of attributes in a vector. When a vector has a small L_2 norm, all its elements are small. According to Sharif et al. [18], there is no pixel over modified if perturbations are counted as small L_2 norm, which means that L_2 norm holds a better outcome when measures perceptual distance among L-p norms.

L_infinity norm measures the max value in a vector. In this case, only the highest value perturbation will be counted, no matter the number of modified pixels. Many adversarial attacks, such as FGSM and BIM, adopted this to control the size of perturbations of the adversarial examples. Limited number of undetectable noises on each pixel can add up to a significant divergence in machine learning models' points of view, resulting in perceptually comparable adversarial attacks.

L_p distances are distances measured by L_p norms, which will be presented between two vectors like:

$$\text{L\_p} - \text{Distance}(\text{x1}, \text{x2}) = \|\text{x1} - \text{x2}\|_p$$

- **Transferability of adversarial attacks**

One of the features of adversarial attacks is the transferability. According to Goodfellow et al. [8], the existence of adversarial attacks is the linearity of models. They believed that once an adversarial attack fooled one model successfully, it is much possible to make other models cause misclassified behaviors with the same task. In this case, they also discovered that models frequently agree on adversarial attacks, which means if one model misclassified the image into a class, then other models may similarly misclassify this image into the same class. Another insight of the transferability of adversarial attacks claimed by Kurakin et al [12], they found if the adversarial attack has strong performance, then it has weak transferability. After more exploration and experiments on this situation, they believed that in comparison to weak attacks, those that are difficult to resist are less likely to transfer between models.

The transferability of adversarial attacks is crucial for black-box attack generations. Due to black-box attacks cannot gain the internal structure of the target model, black-box attacks can be trained to a proxy model, which has a similar result as the target one. By doing so, the black-box attack will have a decent performance if they have good attack rate on the proxy model.

2.3 Adversarial attacks categories

- **FGSM (Fast gradient sign method)**

FGSM is a one-step method for white-box attack. The aim of this method is using the gradient of model's loss function to generate the adversarial example [8].

$$x^{adv} = x + \epsilon \cdot \text{sign}(\nabla_x J(f(x), y)).$$

Where x is a seed sample, y is the true label of x, and f(x) is the target model. J is the loss function of target model, x^{adv} is the calculated noise, and ε is a hyperparameter.

However, the magnitude of noises may not be fully controlled and cause more noises since the one-step behaviour.

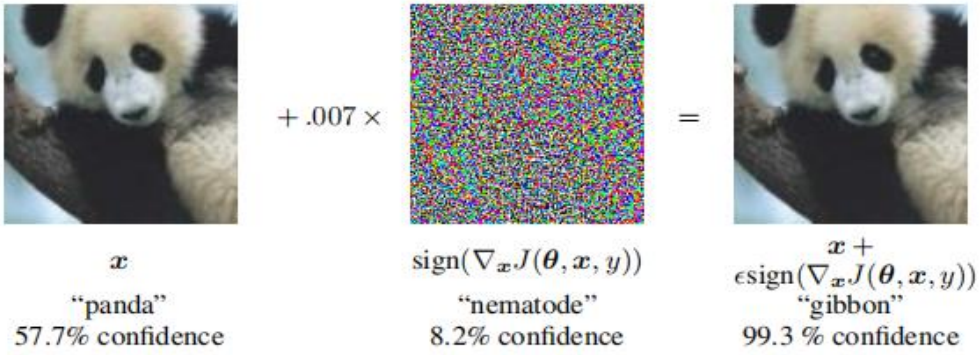


Figure 11. An example of FGSM [8]

- **BIM**

Basic Iterative Method (BIM) is an extension of FGSM, which utilizes the iterative method within the FGSM for improving the attack success rate. Noises are calculated with same equation as FGSM, however, the attacks generated at each iteration will be like:

$$x^{i+1} = Clip_{\varepsilon}(x^{i} + \mu^{i})$$

where the x_i is the range of the seed (from 0 to a predefined value). Clip is the function to ensure the L_p norms distance will lesser than a predefined constraint ε between a generated attack and its seed.

According to Kurakin et al. [12], because of this iterative method, BIM outperforms FGSM, and the resulting attacks have less distortion than FGSM attacks with the identical restriction, for example L_infinity or L_2 norm.

Figure 12. An example of BIM [12]

- **C&W**

C&W, namely the Carlini & Wagner attack algorithm, defined the generation of adversarial attacks as an optimization problem. It aims to cause a misclassification on both targeted and untargeted types, to specify a method to solve the problem [3]. The method idea will be like:

$$\text{Minimize: } \|\mu\|_p + c * f(x + \mu)$$
$$\text{Subject to: } x + \mu \in [0,1]^n$$

Where μ is the perturbation, c is a hyperparameter, and f is an objective function. The c is a constant that to control the importance of attack success rate.

- **Pixel Attack**

Most adversarial attack generations are based on the adversarial examples, which is changing the pixel values of the original images. According to Kotyan and Vargas [11], few-pixel (L_0) attack and threshold (L_∞) attack can affect the adversarial robustness issues on model dependence, insufficient evaluation, false adversarial samples, and perturbation dependent result. In this case, we should pay more attention to the pixel attacks insight for future works. I will introduce and explain more details on this field later.

- **Adversarial Robustness and Regularization**

According to Stutz et al. [19], adversarial attacks can be classified into two categories, namely the off-manifold and on-manifold attacks. Off-manifold, such as FGSM, and on-manifold, such as generative models, will affect different aspects of the model. They proposed that off-manifold attacks can improve the robustness when facing the adversarial perturbations, while on-manifold can help with generalization performance of the model. They focus on exploring a model that can keep both decent robustness and good regularization.

2.4 Related works

- **One Pixel Attack**

According to Su et al. [20], most adversarial attack tactics are adding a large number of well-designed perturbations to create adversarial images, which are believed to be unnoticeable to human eyes. In this case, the classifier may categorize the attacked image as an entirely different class as a result of such modification. However, one significant drawback is the amount of attacked pixels is large, which is hard to achieve the previous goal. Thus, Su's research team proposed a black-box attack that only perturbate one pixel with differential evolution to the original image, and probability labels are the only information needed.

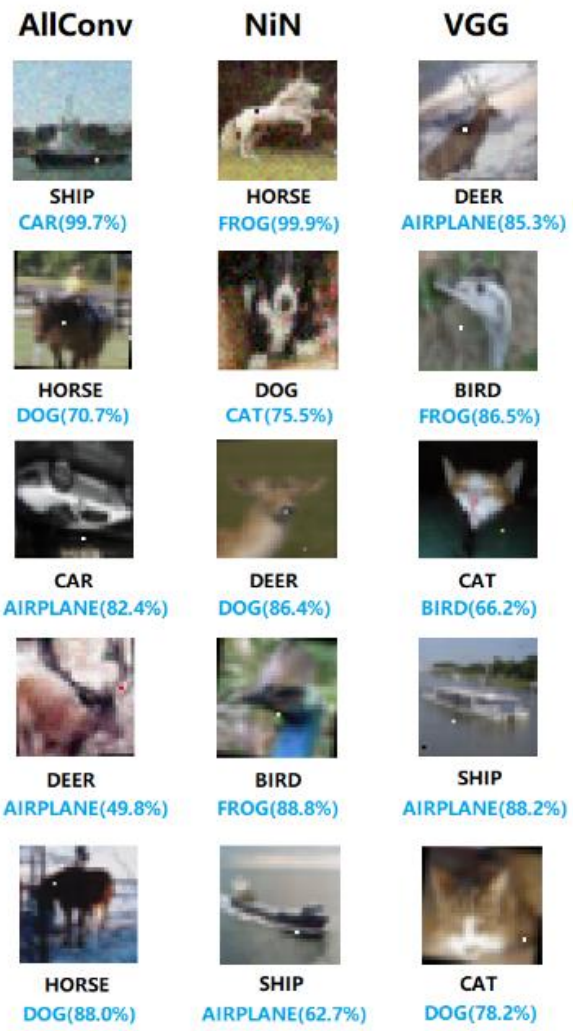


Figure 13. One-pixel attacks created with proposed algorithm; Original image from: "One Pixel Attack for Fooling Deep Neural Networks" [20].

Compared with other adversarial attacks, Su et al. [20] believe that there are three advantages of one-pixel attack. The first advantage is the effectiveness of this attack. By changing one pixel on popular deep neural network structures with CIFAR-10 dataset, it can successfully attack the network with decent result, which means one-pixel attack is approachable. The second feature is one-pixel attack is lean to semi-black-box attack. In this way, there is no inner information such as gradients or specific network structure required for one-pixel attack. This attack will focus on improving the probability label values of the target classes. The last feature is one-pixel attack is flexible that can attack

various deep neural networks, even the network structure's gradient value computation is difficult or is not differentiable [20].

To compare the difference between one-pixel attack and other regular attacks, we assume that an input image could be represented as a vector, with each scalar element representing one pixel. The target classifier f can receive n-dimensional input, $x = (x_1, \ldots, x_n)$ and the true class as t. The probability of x belonging to the true class t, $f_t(x)$. The vector $e(x) = (e_1, \ldots, e_n)$ is the adversarial perturbation related to input x. The target class adv and the maximum modification limit is noted as L. Then, the adversarial problem is transfer to the optimization problem, which is like the following formula:

$$\begin{aligned} \underset{e(\mathbf{x})^*}{\text{maximize}} \quad & f_{adv}(\mathbf{x} + e(\mathbf{x})) \\ \text{subject to} \quad & \|e(\mathbf{x})\| \leq L \end{aligned}$$

In this case, the problem is divided into two parts: which pixel should be attacked and the magnitude of the alteration for each dimension. On the other hand, the one-pixel attack formular is different like below:

$$\begin{aligned} \underset{e(\mathbf{x})^*}{\text{maximize}} \quad & f_{adv}(\mathbf{x} + e(\mathbf{x})) \\ \text{subject to} \quad & \|e(\mathbf{x})\|_0 \leq d, \end{aligned}$$

As we can notice, the d is the limitation number, and d= 1 for the one-pixel attack.

- **Differential Evolution**

According to Das et al. [7], differential evolution is a population-based optimization approach for handling difficult multi-modal optimization issues, which belongs to the general class of evolutionary algorithms (EA). Moreover, it has process in the population selection phase that maintain diversity, such that in reality it is projected to efficiently identify higher quality solutions than gradient-based solutions or even other types of evolutionary algorithms [6]. Specifically, at each iteration, a new set of candidate solutions (children) is generated based on the present population (parents). The

offspring are then compared to their matching parents, with the children surviving if they are more suited, have a higher fitness rating, than their parents. Only by comparing the parent and his offspring can the goal of maintaining diversity and boosting fitness values be achieved at the same time.

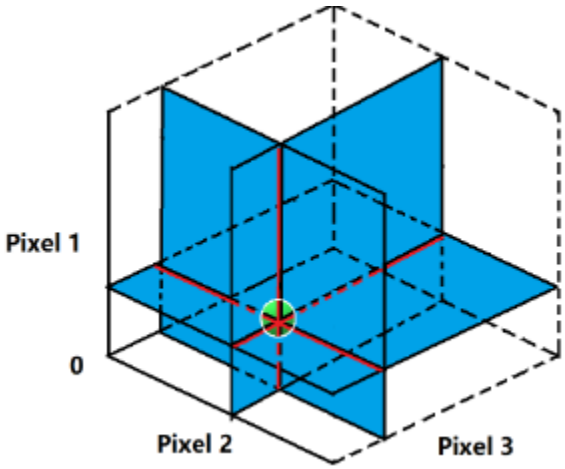


Figure 14. An illustration of using one pixel perturbation attack in a 3-dimensional input space; Original image from: One Pixel Attack for Fooling Deep Neural Networks" [20].

The reason Su et al. [20] choose differential evolution as the core algorithm for their one-pixel attack is DE is not required gradient information for optimizing. Therefore, it can be used to a broader spectrum of optimization problems than gradient-based methods. Also, there are three main advantages of utilizing this algorithm:

a. **Higher probability of Finding Global Optima**: DE is a meta-heuristic that, unlike gradient descent or greedy search algorithms, is less susceptible to local minima. Furthermore, the subject under consideration in this article has a strong constraint (just one pixel can be updated), making it significantly more difficult.
b. **Less Information is needed:** DE does not require a differentiable optimization problem, as do classical optimization methods such as gradient descent and quasi-newton methods. This is important when creating adversarial images because some networks are not differentiable and calculating gradient

necessitates a great deal of information on the target system, which is sometimes unrealistic in many cases.

c. **Simplicity:** The method provided here is unaffected by the classifier employed. Knowing the likelihood labels is all that is required for the attack to take place.

This is also will be the core algorithm I experiment with my proposed one-pixel attacks.

- **Process of Differential Evolution**

There are three stages for differential evolution running with one-pixel attack. The first stage is initializing the candidates, which is the population of the starting point for searching the solution. After initialized candidates, selecting the useful candidates and regenerate them. Last stage is testing these regenerated candidates' value whether suitable for the attack goal, abandon the useless groups and substitute the useful groups. Repeating the stage 2 and stage 3 until it found the higher probability of global optima.

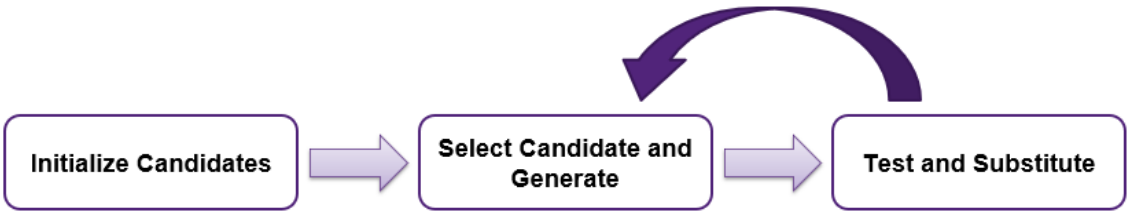


Figure 15. An illustration three stages of processing the differential evolution in one-pixel attack.

There is a simple example of showing the above procedures:

Stage 1: There are five 2-dimension vectors (0 to 1), into $f(x) = \sum x^2/2$:

a. [0.1 0.1] => 1/50

b. [0.2 0.2] => 4/50

c. [0.3 0.3] => 9/50

d. [0.4 0.4] => 16/50

e. [0.5 0.5] => 25/50

Step 2,3: Choose the first vector [0.1 0.1] as the target vector , and choose another 3 vectors: b, c, d

Mutation vector(mut) = 0.5, and do mutant = b + mut * (d-c),

then we have a new vector [0.7, 0.7]

However, the new vector [0.7 0.7] is larger than the target [0.1 0.1], so we will keep the previous target vector.

The goal is to find the minimal vectors of our candidates.

- **Settings for One Pixel Attack**

From the research of Su et al [20], they encode the perturbation into array list, which is optimized by differential evolution. The candidate method consists of a set number of perturbations, and each perturbation is a tuple with five elements: the perturbation's x-y coordinates and RGB values, and one pixel is altered by a single disturbance. The initial population of candidate solution is 400, and after each iteration, another 400 candidate solutions are produced using the standard differential evolution formula as shown below:

$$x_i(g+1) = x_{r1}(g) + F(x_{r2}(g) - x_{r3}(g)),$$
$$r1 \neq r2 \neq r3,$$

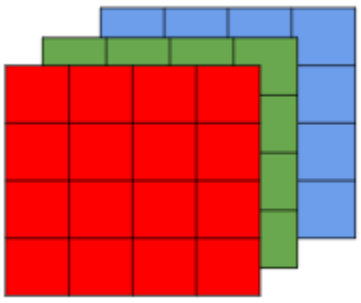


Figure 16. The example of a tuple with five elements: x, y coordinates and RGB values.

Where x_i is an element of the candidate solution, r1, r2, r2 are random numbers. The scale parameter F set to be 0.5, and current index of generation is g. During the generation process, each candidate solution compares with their corresponding parents based on the population index, with the winner surviving to the next iteration. The maximum number of iterations is 100 and early-stop decision criterion for untargeted attack is when the label of true class probability lower than 5%. As for the initial population selection, it is utilizing uniform distributions for CIFAR-10 images, which the image has a size of 32X32. The fitness function is chosen the probabilistic label of the class in the case of CIFAR-10 dataset.

- **Crossover for One Pixel Attack**

The crossover function seeks to improve current solutions by merging them. While single point crossover tends to avoid splitting formal patterns, uniform crossover is more exhaustive since they examine more space and risk breaking good solutions. Among the differential evolution, the crossover function is a decision criterion whether keep searching the global optima until reaches the maximum iteration number. The performance of the crossover processes varies depending on the situation. Crossover offers more variability to the search process than mutation. While crossover generates new methods by splitting and joining old solutions, mutation randomly alters genes.

- **Fitness for One Pixel Attack**

Fitness function is also adopted by differential evolution, which defines the goal of a search. It indicates which aspects of a solution are favored and which are despised, allowing various goals to be met.

- **Stop criterion for One Pixel Attack**

Stop criterion, which is a decision criterion of differential evolution, determines when this algorithm should stop. For one-pixel attack, max fitness and the amount of generation are popular to use. When using max fitness as the stop criterion, it can make

sure the quality of solution, while the amount of generation as the stop criterion, it can limit the generated numbers from the initial candidates of differential evolution.

- **Optimizing One Pixel Attack**

After the original one-pixel attack is proposed, many research teams consider improving the ability to face more complex issues in real-world. Zhou et al. [26] proposed a new one-pixel attack method that reduce the number of calls to the targeted network. There are two methods for their project to achieve the goal: replacing the original differential evolution algorithm by simulated annealing algorithm; Using Structured Adversarial Attack for better initialization of optimization.

- **Simulated Annealing**

Simulated annealing was a popular derivative-free optimization algorithm proposed by Kirkpatrick et al. [10]. It is inspired by the annealing process in physics, in which a substance is heated to a high temperature and then allowed to cool slowly. In nature, annealing allows materials to achieve a lower energy configuration. Similarly, SA has a temperature parameter that starts off high and steadily decreases with each repetition. At each iteration, a neighbor position is picked surrounding the current solution candidate and assessed using the objective function.

$$\Delta E = f(x^{'}) - f(x)$$

where f(x) is the current solution candidate's function value and f (x′) is the newly picked neighbor's function value The acceptance of the new candidate as the new solution is selected probabilistically as a function of both ΔE and the current temperature value. The likelihood of acceptance in basic SA may be defined as follows:

$$P_a = \begin{cases} 1 & if \ \ \Delta E \leq 0 \\ e^{-\Delta E/T} & if \ \ \Delta E > 0 \end{cases}$$

where Pa is the probability of the neighbor being accepted as the new solution candidate and T is the temperature parameter It is evident that when the temperature parameter increases, the algorithm is more likely to choose candidates with higher objective function values. However, when temperature falls, the likelihood diminishes. In other words, the algorithm promotes exploration at the beginning of the process before transitioning to exploitation when the temperature parameter decays [26].

- **Structured Adversarial Attack**

Xu et al. [22] suggest a unique strategy to adversarial attacks that aims to comprehend the structure of the input image in order to generate meaningful perturbations. To detect essential structures, this approach employs a sliding mask across the pixels of the input image. The loss function is then optimized for the hostile inputs across numerous iterations to yield interpretable result.

Figure 17. StrAttack Perturbations Generated example; Original image from: "Optimizing One-pixel Black-box Adversarial Attacks". [26]

To develop a workable attack strategy, Zhou et al.'s [26] preliminary experiments are carried out with StrAttack as the primary adversarial attack. An individual using StrAttack produces good results while decreasing network accuracy. Images from a basic StrAttack implementation, as shown in Figure 14, reveal that the perturbations generated are of modest magnitude and indistinguishable to the human eye, substantially lowering the learning model's accuracy. However, perturbations are generated over a larger number of pixels than the span of their One-pixel Attack tests. Therefore, those perturbations are extracted and using a threshold obtained with empirical methods, which is a binary mask over RGM levels generate every image.

Figure 18. StrAttack Mask across RGB level channels

To compare performance, this mask is used as an initialization to the One-pixel attack. Specifically, they select n pixels at random where the binary mask is positive. The colors for those pixels are then determined as the inverse of the image values. These pixels serve as the initial perturbations upon which the SA optimization method operates [26]. Finally, Figure 16 illustrates a distorted image attacked by their few pixels attack strategy, which successfully caused the model to anticipate the wrong class. The pixel modifications are highlighted in red.

Figure 19. Image after a few pixels attack; Original image from: "Optimizing One-pixel Black-box Adversarial Attacks" [26].

- **Other Materials Reviews**

Senzaki et al. [17] proposed a simple black-box adversarial examples generation method. In their works, they review the adversarial examples generation and introduce a new classification of black-box attacks with few queries. The researchers use datasets, such as MNIST and CIFAR 10, to test a new method for generating adversarial examples. Their report aims to show the effectiveness of their new approach to evade the black-box attacks. This report is useful to my research since authors test their classification by using the MNIST and CIFAR 10 datasets, which will be my test datasets in my thesis project. I can gain some hints about the result of their works. However, there is one limitation is that authors only consider "the adversary can obtain modified outputs of the target classifier" scenario, namely the untargeted one. Thus, the effectiveness of the model needs to prove when deal with the targeted attacks. Another drawback, which is also mentioned in authors report, is their new classification of black-box attacks need to focus on non-domain-specific techniques, which means this new attack method will not lead to the essential solution of the problems. Thus, I need to consider the general solutions or countermeasures for the adversarial attacks when I explore my project topic, instead of one special situation within the attacks.

Alpaydin [1] introduces the concept of machine learning, which is readable and concise as a guidebook for computer science students. The author extends the new content and knowledge, in both theory and application, as the most recent advance in machine learning. It aims to be a compelling textbook for introducing courses in machine learning at both undergraduate and graduate levels. This new edition is useful to my research topics because the author adds a new section on generative adversarial networks, which will help me with knowing the concept of the relationship between AI models and adversarial methods. The main limitation is this book focuses on introducing the overall background of machine learning, instead of a specific area related to my thesis project. Therefore, this book only provides a huge range of machine learning background knowledge in general.

By reading the article, Goodfellow et al. [8] review the adversarial examples by explaining the nonlinearity and overfitting problems. The authors explain the linear perturbation and non-linear model by using classic adversarial examples. Also, the authors use the MNIST dataset for approaching the maxout network by changing the test set error. One useful information to my research is this article explain the adversarial examples on comparison of linear and non-linear model and using the MNIST dataset. One limitation of the article is the gradient-based optimization of modern AI still need to be proved with variety methods. However, the authors have proved that the development of optimization stages can train the models, which is contain a good stability. Although this article will not be the basis of my research, it gives me some potential ideas of evading adversarial examples on linear and non-linear models. In addition, Kurakin et al. [12] propose some adversarial examples are malicious factors to attack and fool machine learning and AI models. The authors design some adversarial training such as changing the training to large models and dataset, observing the robustness by using single-step attack methods, and figuring out the differences and disparity between single-step and multi-step attacks for mounting black-box models. They focus on finding an advanced way to increase robustness to deal with adversarial examples. The main limitation is the dataset for their experiments is lack of universality. In this case, the result gives me one possible idea of training the AI model by using the single-step attack to evade the potential risks. By learning the technique, it will be useful supplementary method for improving the robustness of the AI models.

As for the adversarial example generations and attacks, some reports and articles give me some useful hints for my experiment works. Said et al. [16] introduce an adversarial attack tactics can evade some countermeasures at the AI hardware accelerator stage, while it will be activated by rogue intentions. The authors use some deep learning models, such as kernel Conv2D function and Verilog RTL model, to present the accelerator-level universal adversarial noise attack under the FuseSoC environment. The research focuses on adversarial attacks on AI hardware systems. The article is useful since it provides a comparison of adversarial noises effects between

hardware systems and software systems. However, this will be the limitation that the output may be different based on hardware divergence. This article expands my horizons when thinking about not only the AI software system can be attacked. Moreover, Sun et al. [21] generate the threat model for attacks by using adversarial examples and analyze the limitations and core procedures to deal with adversarial example attacks in the real world. The authors use the notion of adversarial example, threat models, and attack routines to handle some existing practical attacks such as eyeglasses frame, road sign, and infrared. Their research focuses on evading the potential adversarial attacks on systems with AI in the future. The article is beneficial to my thesis since it shows the basic logic for the adversarial example attacks and some inspirations for designing the models. The main drawback is the research lack of some test procedures with big datasets; therefore, the authors can only provide some survey results on mounting methods and detection methods. Furthermore, Zhang et al. [25] introduce and explain the logic of adversarial feature selection to against evasion attacks. The authors offer a specific investigation of shedding some light on the security properties of feature selection against evasion attacks. They focus on validating the soundness on various wrapper-based implementation when faced with adversarial examples. This article gives me one hint that considering the wrapper-based model when dealing with the adversarial attacks. However, one drawback of this article is the traditional feature selection methods' performance may be influenced by different adversarial attacks, thus this may cause difficulty when training the model to handle various types of attacks. The idea of feature selection and wrapper-based model may be one potential aspect to improve some black-box attacks.

Furthermore, some defense tactics and have been provided by Lal et al.'s works [13]. They approach a defensive model to against the adversarial attacks, adversarial training, and a feature fusion strategy with validated labels to design the classification. The authors design the DR classification by using the original and perturbed images to run different adversarial attacks, such as FGSM, then use these datasets to test adversarial attack generations. Their research focuses on improving the model to understand the image

processing techniques and mitigate the influence of adversarial attacks. There is useful factor that can contribute to my thesis project is the authors propose the adversarial training-based defense, which provides me a hint for improving the evade of the attacks. The main limitation is it only be tested with convolutional neural network model, which is can only offer the result of using this method. But the procedures of designing the defensive model with CNN method can give me a potential way to complete my project. Another good example proposed by Ming et al. [14]. They propose an adversarial attack to learning model, which depends on autoencoders' latent representation obtain. The authors use the MNIST and CIFAR-10 dataset with an instance configuration to illustrate their designed model based on semantic encoder. They focus on the potential value of training the adversarial attacks for their model. It is useful to my research topic since the authors suggest that the network architecture of the semantic autoencoder can show a decent performance when dealing with the adversarial attacks. One drawback of this proposed semantic autoencoder is the cost performance when using t-SNE within the model's architecture. However, I can realize many experiences by handling the adversarial attacks with devised model.

3 Proposed Works

The goal of this project is experimenting both white-box and black-box attack generations to several machine learning models, then explore the insight of these attacks and find some methods to improve the original adversarial attacks. In this case, I will pay more attention to improve the original one-pixel attacks. To achieve this, there are two tasks:

Firstly, I will test two popular white-box attacks: FGSM (Fast gradient sign method) and BIM (Basic Iterative Method), to explore the insight of adversarial attacks. Secondly, after I get some findings from the previous white-box attack generations, the main job for this project is to find several solutions that can improve the original one-pixel attack, (black-box) either increase the success rate of the attack or shorten the time of the attack. Generally, the idea of adversarial training is injecting adversarial examples into the training datasets. Thus, I will use two popular datasets: MNIST and CIFAR-10. And white-box attacks experiments will mainly utilize the MNIST dataset because it contains black and white (bilevel) handwritten digits to fit in a 20X20 pixel box when preserving their aspect ratio and then translated into the center of 28X28 field. It is a decent dataset that for testing some user-friendly level machine learning model and adversarial attack algorithms. Then test FGSM and BIM with CIFAR-10 dataset, which these two results can be the reference for one-pixel attack's result with the same dataset. As for the black-box attacks, especially the original one-pixel attack and my proposed one-pixel attack, I will use CIFAR-10 as the main dataset.

Figure 20. Example of MNIST and CIFAR-10 dataset

- **3.1 White-box attacks**

- **FGSM (Fast gradient sign method)**

As mentioned before, FGSM was proposed by Goodfellow et al. [8]. To be specific, instead of working on how to minimize the loss by adjusting the weights referred to gradients, this attack aims to adjust the input data to maximize the loss.

Algorithm 1 Adversarial training of network N.
Size of the training minibatch is m. Number of adversarial images in the minibatch is k.

1: Randomly initialize network N
2: **repeat**
3: Read minibatch $B = \{X^1, \ldots, X^m\}$ from training set
4: Generate k adversarial examples $\{X^1_{adv}, \ldots, X^k_{adv}\}$ from corresponding clean examples $\{X^1, \ldots, X^k\}$ using current state of the network N
5: Make new minibatch $B' = \{X^1_{adv}, \ldots, X^k_{adv}, X^{k+1}, \ldots, X^m\}$
6: Do one training step of network N using minibatch B'
7: **until** training converged

Figure 21. Normal adversarial training algorithm example

There are three important inputs:

- Epsilons: A list of epsilon values to use for the attack run. The data range is [0, 1], so epsilon value should not exceed 1. We will try different epsilon values to figure out the differences.
- Pretrained-model: Since the model is not the main issue for our project, using the pretrained dataset model provide us a simple way to test the attack (e.g. Lenet-5)
- Cuda: This is a Boolean flag to test with GPU or CPU. We will use GPU to test attack.

Then I will follow the FGSM function:

$$x^{adv} = x + \epsilon \cdot \text{sign}(\nabla_x J(f(x), y)).$$

where the output will be the perturbed image. Three inputs include: original image (x), pixel-wise perturbation amount (ϵ), and the gradient of the loss function the input image.

Then, I will try to compute the result with a test function. The function will perform a full test step on the MNIST dataset and run out the accuracy by changing the epsilon values such as 0, 0.1, 0.15, 0.2, 0.25, 0.5. After finishing that process, I could illustrate an accuracy versus epsilon plot by using the result. Then, I will choose the most suitable result's epsilon value for the next white-box attack.

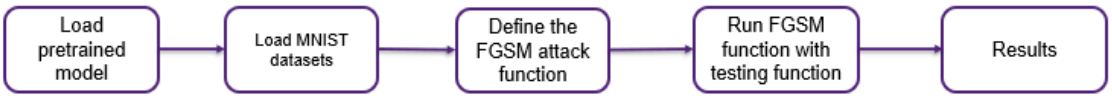


Figure 22. FGSM experiment workflow

- **BIM (Basic Iterative Method)**

After I got the result of FGSM, the next step is testing the BIM attack with the same dataset (MNIST) and the same model (Lenet-5). This method is similar to FGSM's since it

is an extension of FGSM but applies FGSM multiple times with small step size. The Clipx,ϵ denotes clipping of input in the range of [X - ϵ, X + ϵ], we can adjust the parameter to approach the most suitable one.

$$X_0^{adv} = X$$
$$X_{N+1}^{adv} = Clip_{X,\,\epsilon}\left(X_N^{adv} + \alpha\, sign\left(\nabla_X J(X_N^{adv}, Y_{true}) \right) \right)$$

There are three important inputs:

- Epsilons: A list of epsilon values to use for the attack run. The data range is [0, 1], so epsilon value should not exceed 1, which is the same constrain as FGSM's. To compare the performance of these two white-box attacks, I will use the most suitable epsilon value from the previous FGSM experiment, and then use the same one for BIM.
- Pretrained-model: Using the pretrained dataset model provide us a simple way to test the attack (e.g. Lenet-5)
- Num_step: The number of small step times. This is the main difference between FGSM and BIM. The number represents how many iterations will perform. According to the configuration of my computer and the actual situation, I will choose an appropriate value for this. Although the larger value of iteration should show better results, it will take much attacking time, and the result will not get better since the attack performance has reached a critical point.

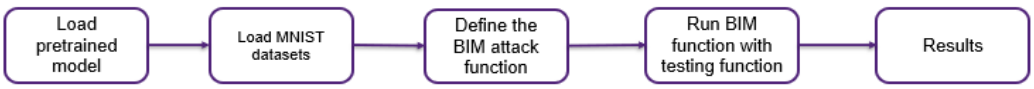


Figure 23. BIM experiment workflow

- **3.2 Black-box attacks**

- **improved one-pixel attack**

As mentioned from the Section 2, Su et al. [20] proposed this black-box attack by perturbating only one pixel of the original image in a scenario where only the probability labels are available for attackers. Compared to the FGSM and BIM attacks, which the perturbation will be a large number of pixel values, one-pixel attack's perturbation will only have one pixel value (Figure 20).

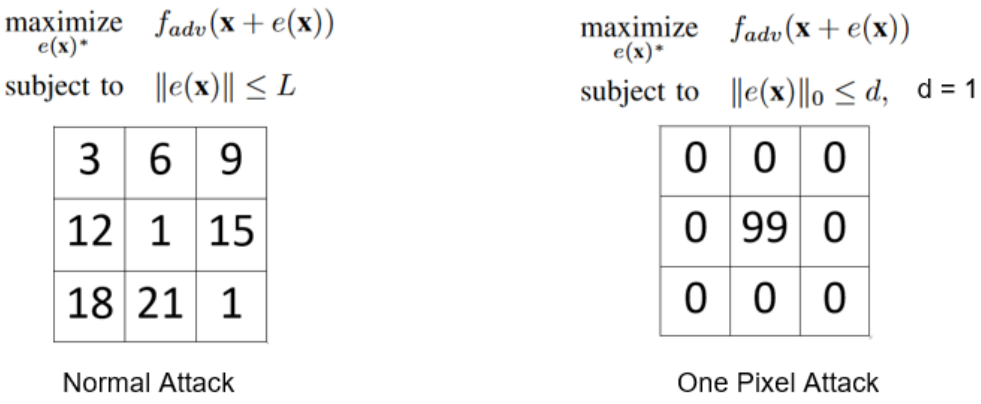


Figure 24. Comparison of the normal attack's perturbation and one-pixel attack's

The original size of initializing candidates (population) for each iteration is 400, the decision criteria when keep the candidate is the true probability of attacked images is under 5%, and there is no restriction of stop and generates the new candidates until the last iteration complete.

My idea of improving the one-pixel attack is:

- Change the size of initial candidates for each iteration
- Change the decision criteria for keeping the candidate
- Set a restriction of stop and generates the new candidates until the last iteration complete

There are two factors for evaluating the proposed attack performance:

- Attack success rate: how confident is the attack will succeed
- Attack time: how much time does the attack take

To achieve this goal, I will test and get the result of the original one-pixel attack first. Then, I will follow the steps below:

1. Experiment only changes the size of initializing candidates for each iteration:

100, 200, and 600

2. Experiment only changes the decision criteria when attack the image:

True probability (fitness) of attacked image is under 15%, and 25% with early stop restriction (over 80%)

3. Experiment changes the size of initializing candidates for each iteration and the decision criteria together:
   - Candidate population = 100 and fitness < 0.15
   - Candidate population = 100 and fitness < 0.25
   - Candidate population = 200 and fitness < 0.15
   - Candidate population = 200 and fitness < 0.25
   - Candidate population = 600 and fitness < 0.15
   - Candidate population = 600 and fitness < 0.25

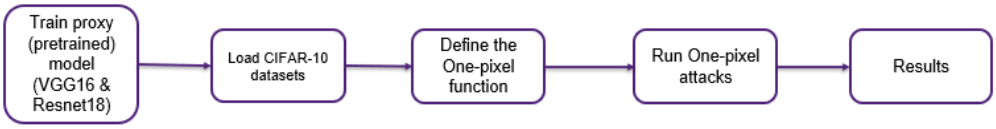


Figure 25. One-pixel attack experiment workflow

4 Experiments and Results

I implemented my test by using Python in PyTorch environment and did experiments on MNIST and CIFAR-10 dataset.

- **Experiment environments:**

All the experiments are tested on my desktop:

- **CPU**: Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz , 2.90 GHz
- **GPU**: Geforce GTX 1660 Ti
- **Language**: Python
- **Major machine learning library**: Pytorch, Adversarial Robustness Toolbox (ART)
- **Dataset**: MNIST, CIFAR-10
- **Auxiliary tools**: Word, Lucidchart, Excel

4.1 White-box attacks

- **FGSM & BIM on MNIST**

First white-box attack experiment is the FGSM. Since the model is not the main improved attribute for attacks, I used the pretrained model (LeNet-5) from the Pytorch library which was trained with MNIST dataset. The example LeNet-5 model structure is shown below (Figure 26):

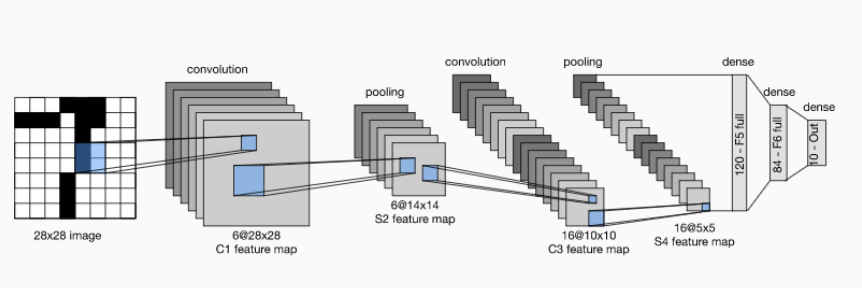


Figure 26. Example of data flow in LeNet-5

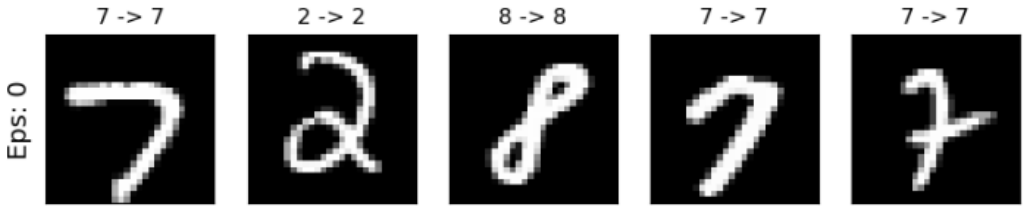
Before I define the FGSM attack function, I load the MNIST dataset by using the DataLoader from PyTorch library and divided the MNIST dataset into two part: 50,000 training images and 10,000 testing images. Then, I define the function for creating the FGSM adversarial examples by perturbing the original inputs. There are three inputs: the original image (x), the epsilon value, which represents the perturbation amount (ϵ), and the gradient of the loss function for the input image ($\nabla_x J(\theta,\mathbf{x},y)$). Then the function will create adversarial image as we mentioned in Section 2:

*Adversarial image = image + epsilon*sign (gradient of the loss)*

$$= x + \epsilon * sign\left(\nabla_x J\left(\theta, \mathbf{x}, y\right)\right)$$

to ensure the range of the data is not exceeded, the adversarial image will be clipped to range [0, 1].

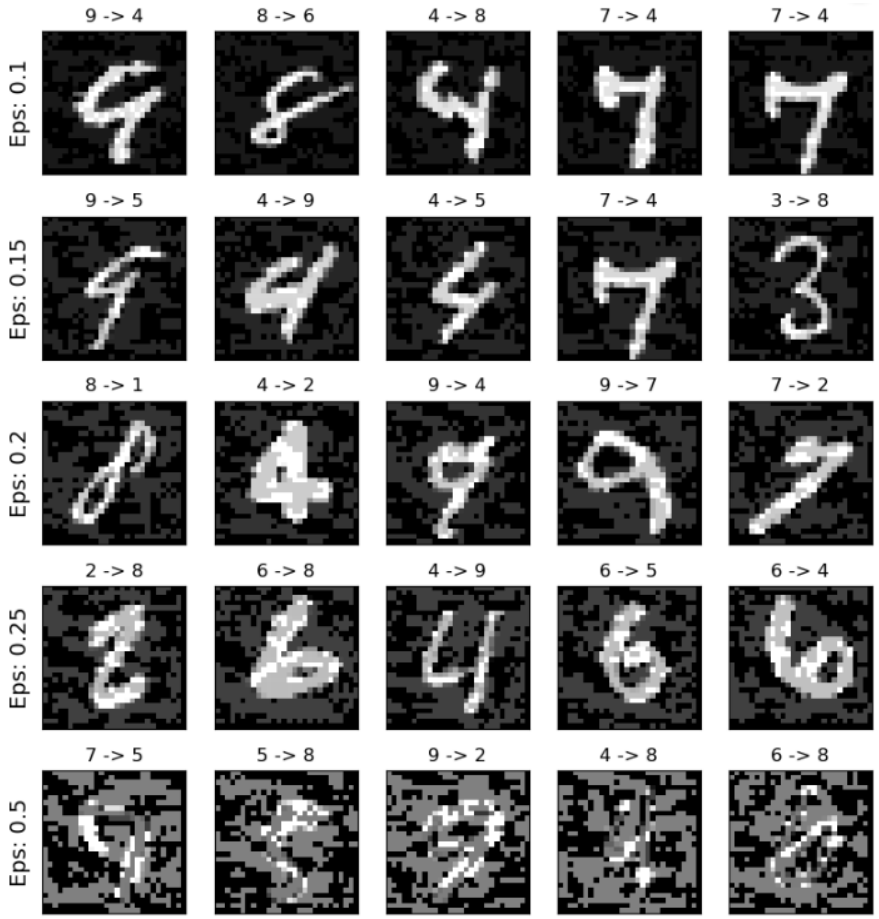
After the attack function, the testing function shows the test steps on the MNIST test set and reports the final accuracy. Specifically, for each sample in the test set, the function computes the gradient of the loss with respect to the input data (gradient of data), produces a perturbed image with FGSM attack, and then checks to see if the perturbed example is adversarial. In addition to assessing the model's accuracy, the function stores and delivers some successful adversarial samples to be displayed.




```
Epsilon: 0      Test Accuracy = 9810 / 10000 = 0.981
```

Figure 27. The result of the LeNet-5 model without FGSM attack

The accuracy of the LeNet-5 model has a great performance, with around 98% to classifier the input into the correct class. Then, the result of FGSM with various epsilon values will shown below:




```
Epsilon: 0.1    Test Accuracy = 8510 / 10000 = 0.851
Epsilon: 0.15   Test Accuracy = 6826 / 10000 = 0.6826
Epsilon: 0.2    Test Accuracy = 4301 / 10000 = 0.4301
Epsilon: 0.25   Test Accuracy = 2082 / 10000 = 0.2082
Epsilon: 0.5    Test Accuracy = 63 / 10000 = 0.0063
```

Figure 28. The result of the LeNet-5 model with FGSM attack, epsilon value: 0.1, 0.15, 0.2, 0.25, 0.5

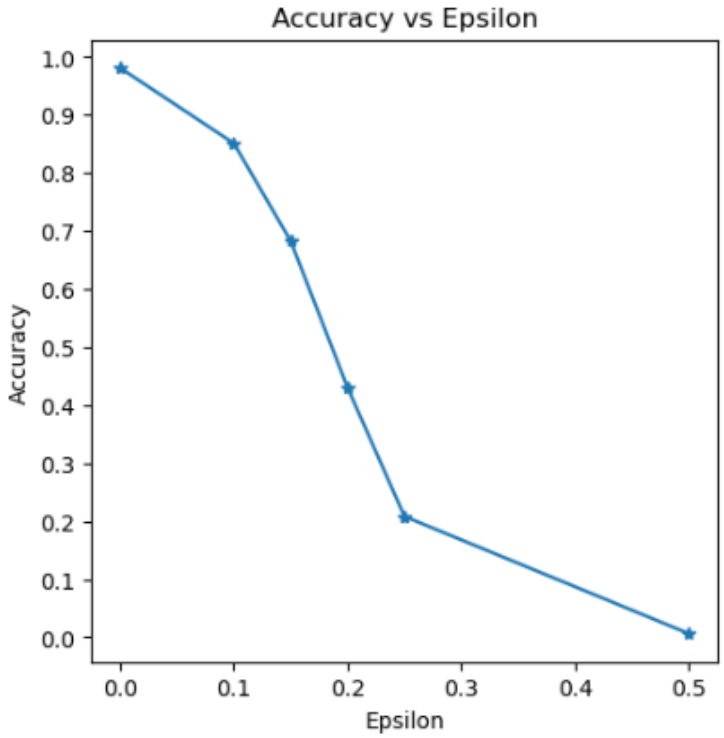


Figure 29. The graph shows the accuracy of different epsilon values

As we can see the result samples from Figure 28 and Figure 29, the larger the value of epsilon, the lower the accuracy of the model's ability to recognize images correctly. However, when epsilon value is too large, such as 0.5 for this experiment, the output image is too easy for the human eyes to figure out that it has been attacked, which will not be useful when facing the real-world problems. However, although the result of epsilon value with 0.1 is hard to be detected by human eyes, the attack rate is kind of low. Another highlight from the figure 24 is when epsilon value larger than 0.25, the accuracy starts to decrease slowly, instead of decreasing sharply as before. This might be the FGSM has reached the limitation of this attacking algorithm. Thus, I think the 0.15 epsilon value is the best one for testing the next white-box attack, BIM. And then we will compare these two white-box attacks performance to seek any insight is useful for our black-box attack in the next stage.

The process of experimenting with BIM is similar to the one with FGSM. Firstly, load the pretrained model (LeNet-5) from Pytorch library. Then, load the training and test MNIST dataset. After that, I decide to use the epsilon value = 0.15 for BIM attack

function, and the num of iteration step is 3. After all functions are settled, the result is shown below:

```
Accuracy on adversarial test examples with BIM: 6.18%
```

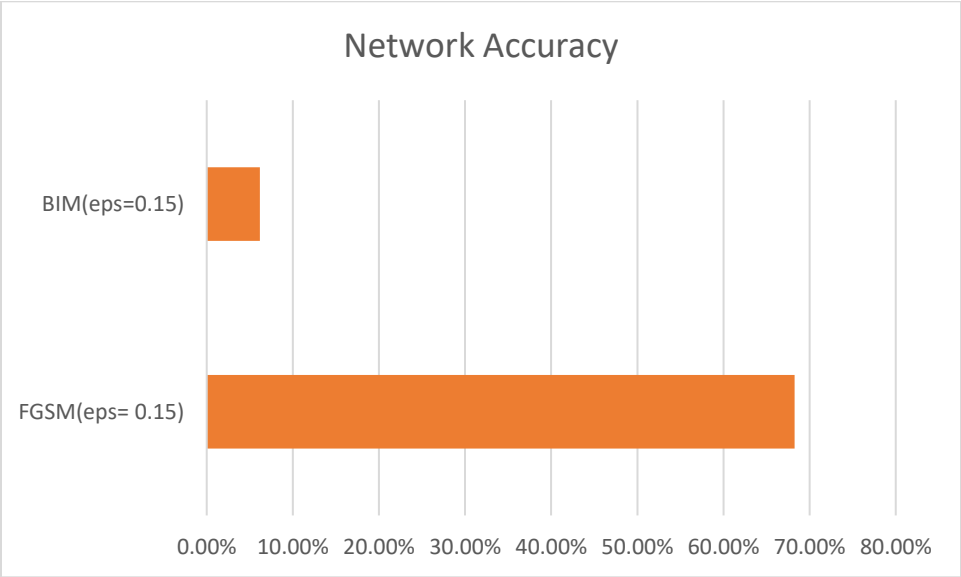


Figure 30. Comparison of BIM and FGSM with the same eps value with same dataset

The result is significant that the extension version of FGSM, which is BIM, can achieve a massive attacking rate after 3 iterations step for each input image. The insight of these two attacks proves that using the same attack algorithm with suitable iteration steps will sharply increase the attacking rate.

- **FGSM & BIM on CIFAR-10**

Testing FGSM and BIM on CIFAR-10 used pretrained VGG16 model form PyTorch library.

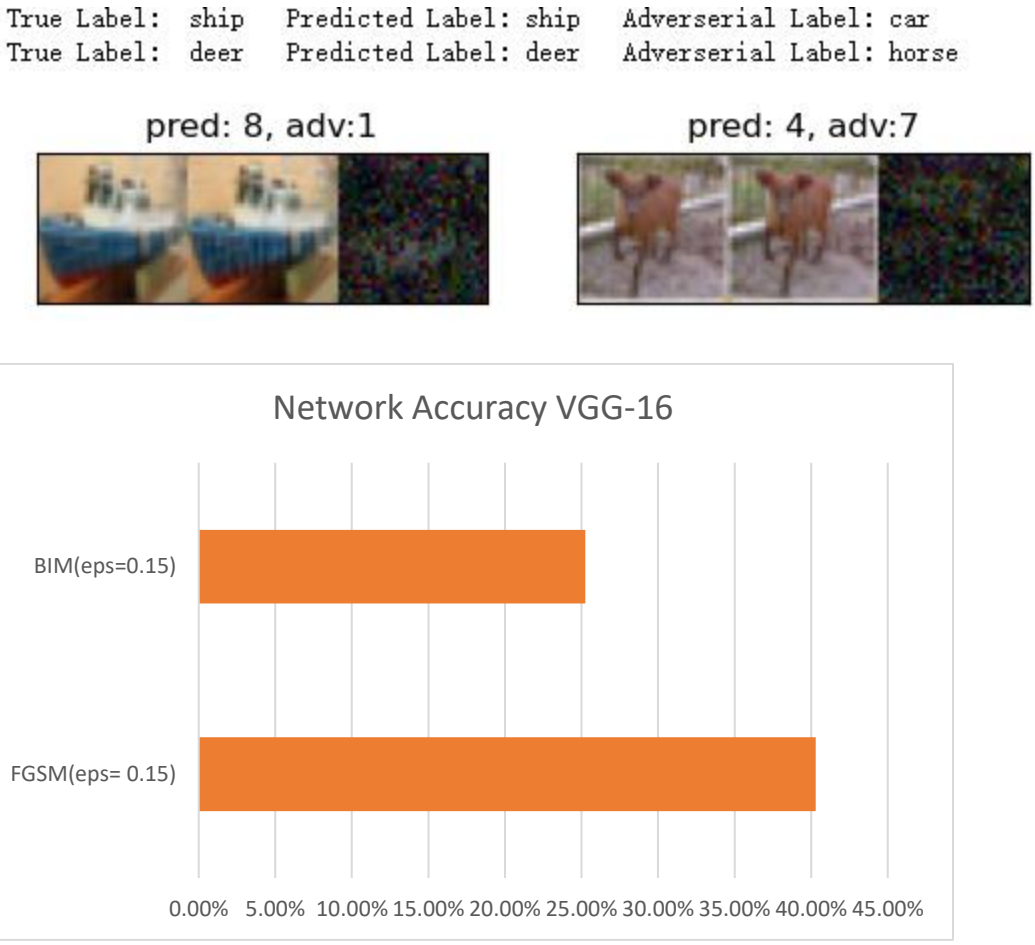


Figure 31. Up: Example of BIM on CIFAR-10; Down: The accuracy of VGG-16 after white-box attacks

From the Figure 31, we get the same result, which the BIM performs better than FGSM with the same dataset (CIFAR-10) and same epsilon value.

4.2 Black-box attacks

- **Original one-pixel attack on CIFAR-10**

The original one-pixel attack proposed by Su et al. [] test three models: All Convolution Network, Network in Network, and VGG16 Network. In my experiment, I utilized VGG16 Network and ResNet-18 Network for exploring my proposed works. These two chosen networks are still pre-trained from PyTorch library for testing CIFAR-10 dataset directly.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 32. Example of VGG networks from PyTorch library

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 33. Example of ResNet networks from PyTorch library

From the previous white-box attack results, the original on-pixel attack, FGSM (eps =0.15), and BIM (eps = 0.15) results of VGG16 models are shown below:

Attack Method/Network	Network Accuracy	
	VGG-16	Resnet18
Without attck	80.77%	80.83%
FGSM(eps= 0.15)	40.29%	
BIM(eps=0.15)	25.25%	
Original OP attack	46.70%	48%

Figure 34. Network accuracy in different situations.

As mentioned above, the candidate solution of differential evolution algorithm is the core task for one-pixel attack. Every candidate result contains a fixed number of perturbations, which contains five elements as a tuple: x, y coordinates and RGB value of the perturbation. The size of random choose children candidate for each iteration: 3. X_1, X_2, X_3, and the mutation equation will be mutant = $X_1 + F * (X_2 – X_3)$, where scale parameter F = 0.5. Maximum iteration number is 75. These conditions will not change for every attack experiment. Then, for the parameters used to test our proposed works, I list them in each experiment and present my results.

The original one-pixel attack's parameters:

- Initial number of candidate population: 400
- Decision Criteria: true probability of attacked image is under 5%
- Number of test samples: 30 & 50

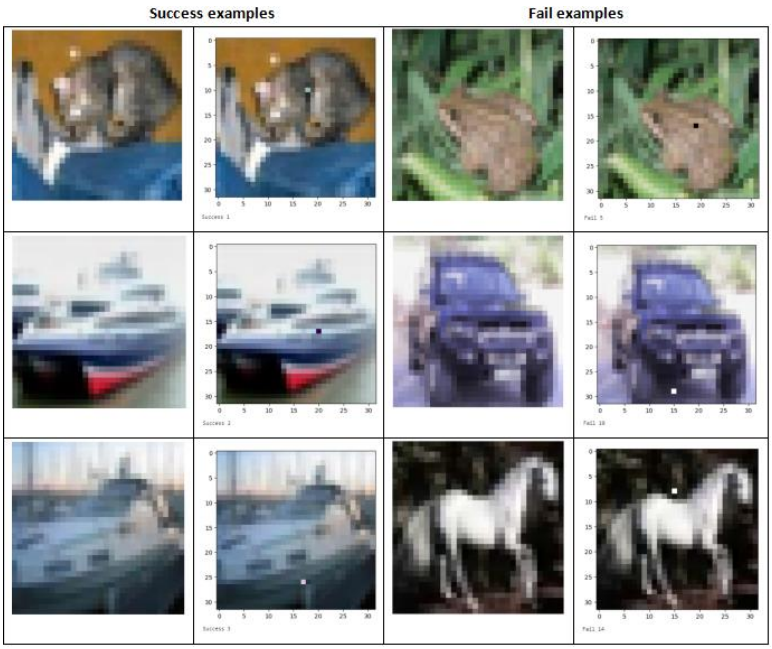
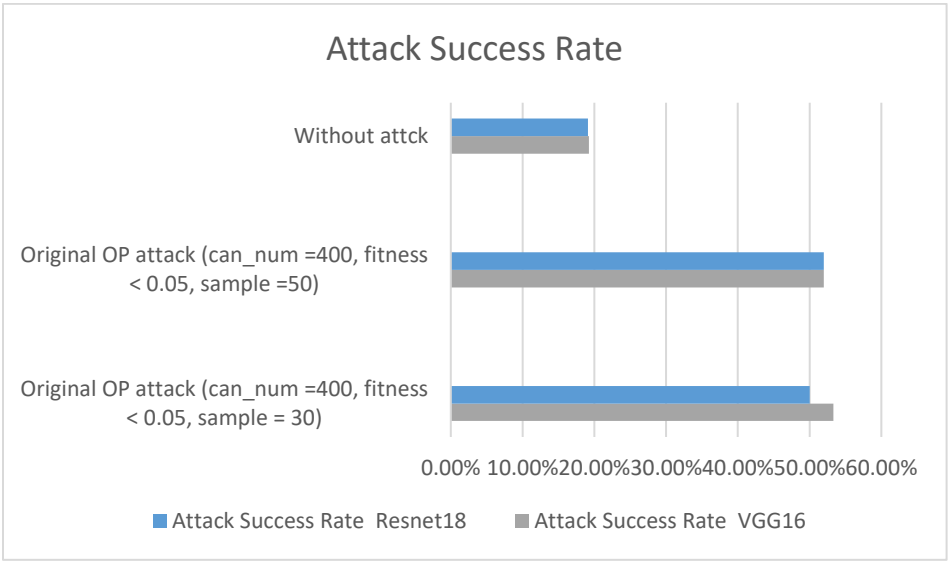


Figure 35. Success and fail examples of one-pixel attacks



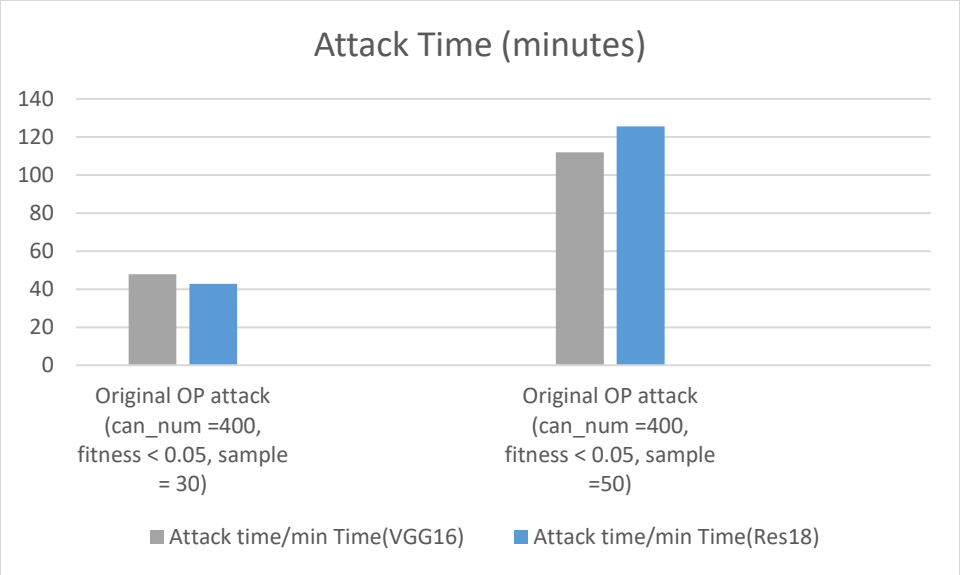


Figure 36. Up: Result of original one-pixel attack in different sample size; Down: Result of attack time

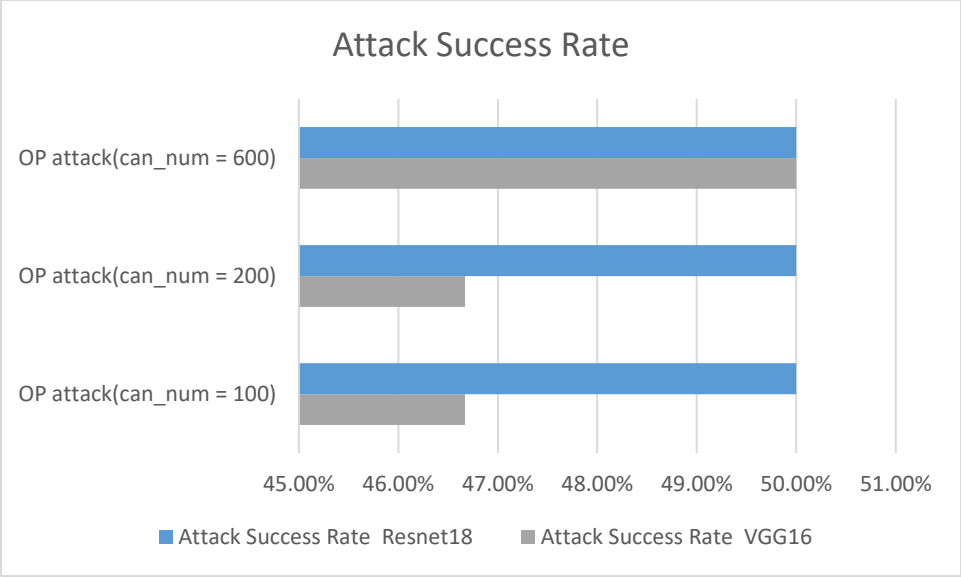
From the Figure 36, we can see that both VGG16 and ResNet-18 have decent success rate of classifying input images correctly without attack, under 20% mistake ratio. However, both will have over 50% chance to cause mistakes when under one-pixel attacks. On the other hand, it takes much more time when the sample size is larger, for example, it takes under 50 minutes when sample size is 30, while it takes over 100 minutes when sample size is 50.

- **My improved one-pixel attack on CIFAR-10**

Next stage, I will experiment with changing candidates' number only, changing decision criteria only, and changing candidates' number and decision criteria together.

The parameters of changing candidates' number only:

- Initial number of candidate population: 100, 200, 600
- Decision Criteria: true probability of attacked image is under 5%
- Number of test samples: 30



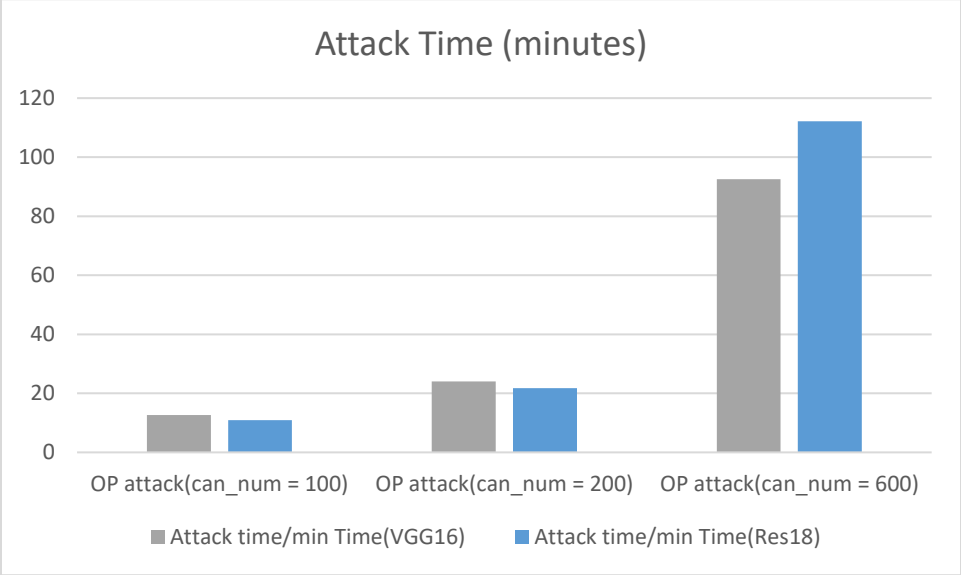


Figure 37. Up: Result of different candidates' number; Down: Result of attack time

From the Figure 37, it seems that larger or small size of candidate does not affect Resnet18 model's performance when sample = 30, but it influences the performance of VGG16. However, it takes too much time when candidates' num is large but result in a similar attack success rate.

The parameters of changing decision criteria only:

- Initial number of candidate population: 400
- Decision Criteria: true probability of attacked image is under 15%, 25%
- Number of test samples: 30

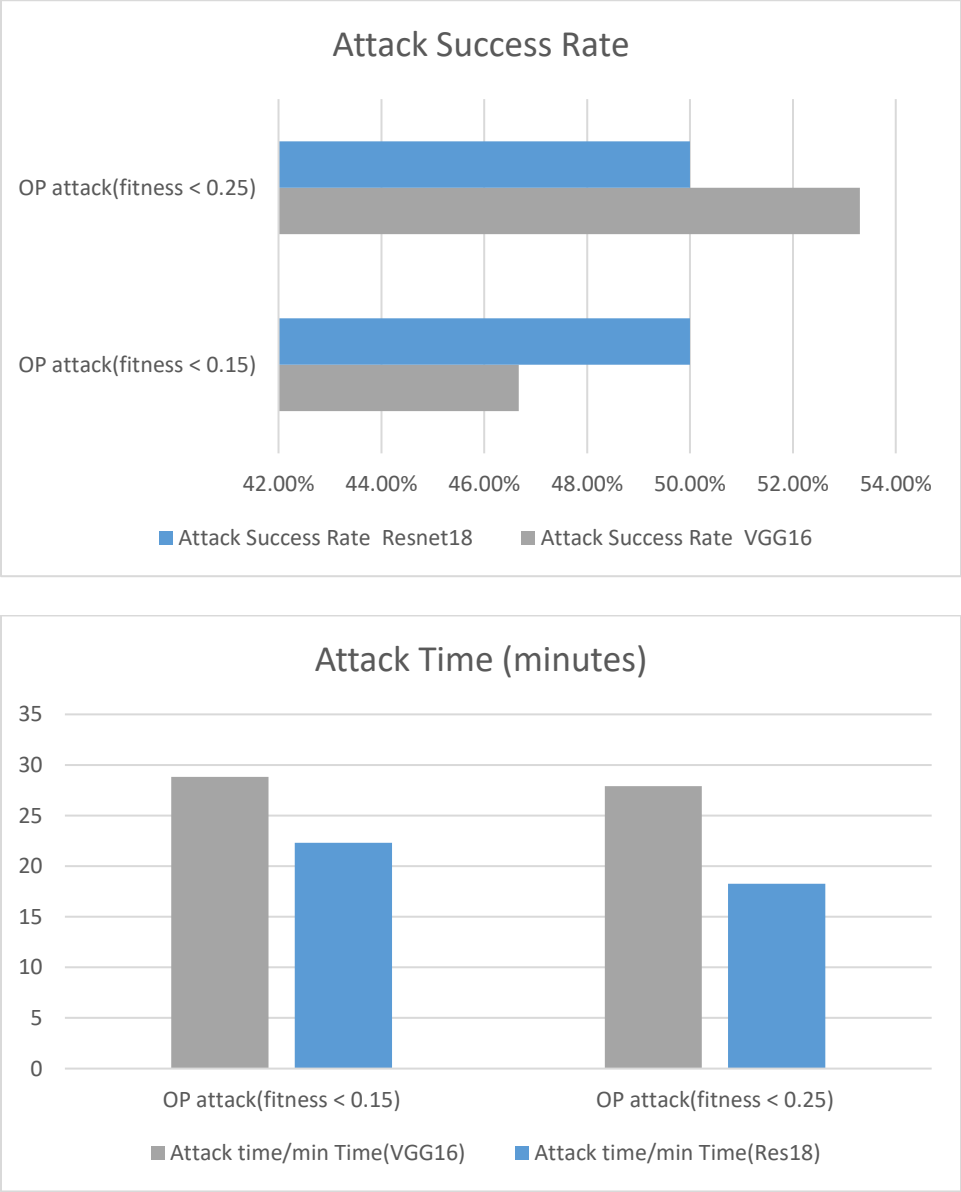
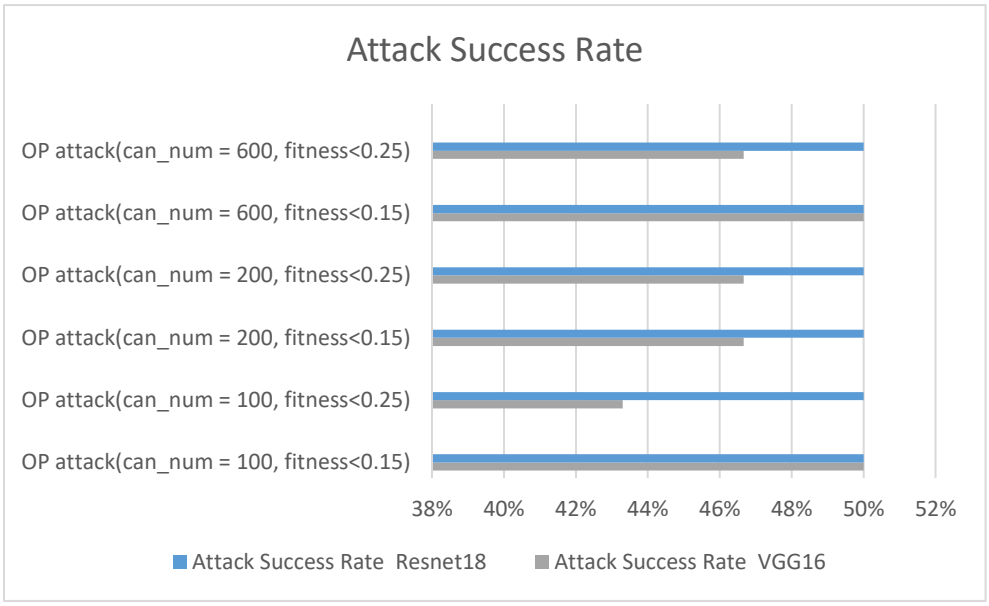


Figure 38. Up: Result of different decision criteria; Down: Result of attack time

From the Figure 38, Resnet18 has the same result of different decision criteria. However, VGG16 performs better when true probability of attacked image is under 25%. Compared to the original one-pixel attack time (Figure 36), changing the decision criteria shorten much time.

The parameters of changing candidates' number and decision criteria together:

- Initial number of candidate population: 100, 200, 600
- Decision Criteria: true probability of attacked image is under 15%, 25%
- Number of test samples: 30



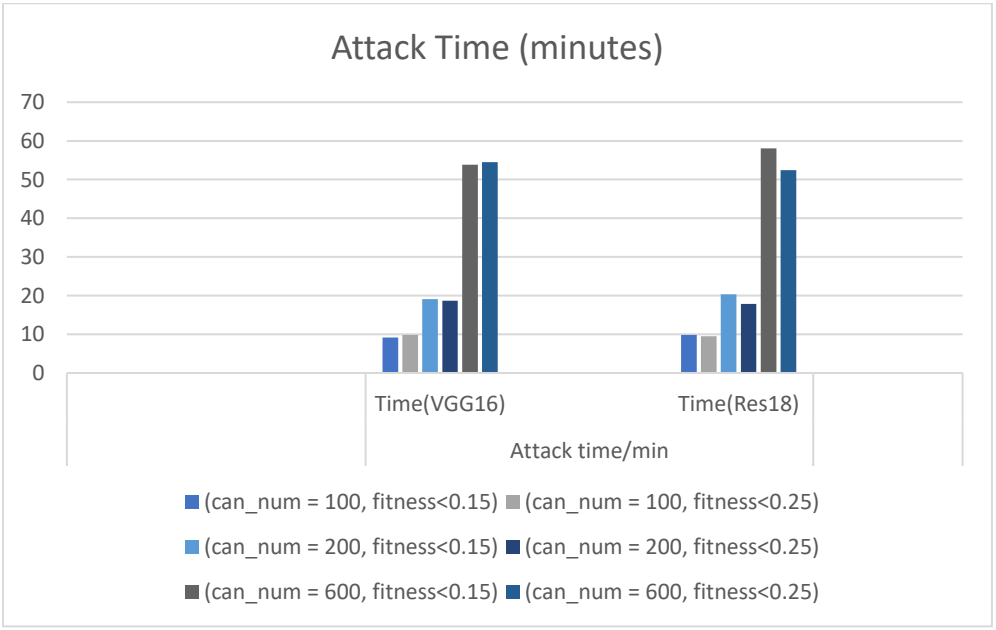


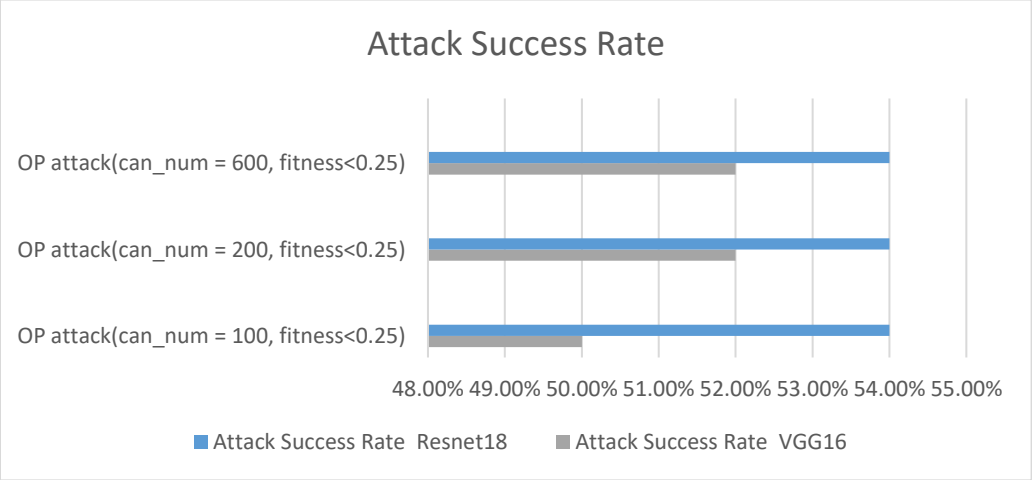
Figure 39. Up: Result of success rate; Down: Result of attack time

From the Figure 39, the attack success rate of all attacks on VGG16 model has dropped, while those on Resnet18 model has remained the same result: 50%. On the other hand, the attack time for all of them has become much shorter than the original one-pixel attack.

Last experiment, I want to explore the extreme situation for a larger number of test samples with my proposed idea. And the result will decide whether my proposed work can achieve a better performance either can increase the attack success rate or shorten the attack time.

The parameters of extreme situation:

- Initial number of candidate population: 100, 200, 600
- Decision Criteria: true probability of attacked image is under 25%
- Number of test samples: 50



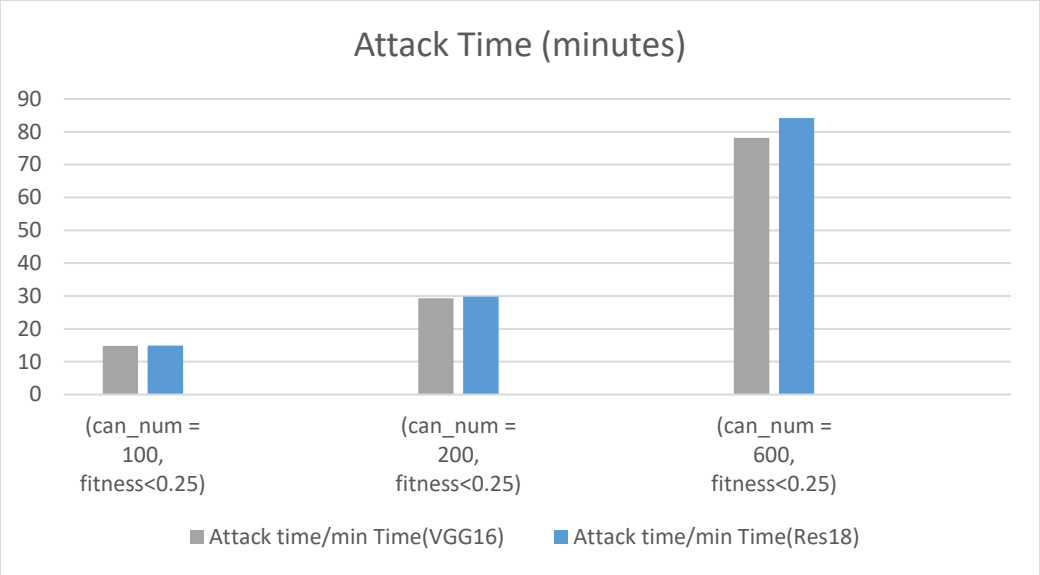


Figure 40. Up: Result of success rate; Down: Result of attack time

From the Figure 40, the attack success rate of Resnet18 in three different extreme situations are still result in a steady performance with 54%, which is better than the original one-pixel attack. On the other hand, VGG16' attack success rates are slightly lower than the original one. But both attacks show less time for them attack the models. The results of all proposed experiments are in the following table.

	Attack Success Rate		Attack time	
Attack Method/Network	VGG16	Resnet18	Time(VGG16)	Time(Res18)
Original OP attack (can_num =400, fitness < 0.05, sample = 30)	53.30%	50%	47m 46s	42m 52s
OP attack(can_num = 100)	46.67%	50%	12m 36s	10m 55s
OP attack(can_num = 200)	46.67%	50%	24m 02s	21m 42s
OP attack(can_num = 600)	50.00%	50%	1h 32m 34s	1h 52m 8s
OP attack(fitness < 0.15)	46.67%	50%	28m 50s	22m 18s
OP attack(fitness < 0.25)	53.30%	50%	27m 55s	18m 15s
OP attack(can_num = 100, fitness<0.15)	50%	50%	9m 12s	9m 50s
OP attack(can_num = 100, fitness<0.25)	43.30%	50%	9m 49s	9m 30s
OP attack(can_num = 200, fitness<0.15)	46.67%	50%	19m 5s	20m 18s
OP attack(can_num = 200, fitness<0.25)	46.67%	50%	18m 41s	17m 49s
OP attack(can_num = 600, fitness<0.15)	50.00%	50%	53m 50s	58m 4s
OP attack(can_num = 600, fitness<0.25)	46.67%	50%	54m 28s	52m 27s

Figure 41. All experiments result (sample size = 30)

Attack Method/Network	Attack Success Rate		Attack time	
	VGG16	Resnet18	Time(VGG16)	Time(Res18)
Original OP attack (can_num =400, fitness < 0.05, sample =50)	52.00%	52%	1h 51m 53s	2h 5m 31s
OP attack(can_num = 100, fitness<0.25)	50.00%	54%	14m 46s	14m 57s
OP attack(can_num = 200, fitness<0.25)	52.00%	54%	29m 17s	29m 51s
OP attack(can_num = 600, fitness<0.25)	52.00%	54%	1h 18m 5s	1h 24m 12s

Figure 42. All experiments result (sample size = 50)

5 Conclusions and Future work

- **Conclusions**

During this project, I tested three adversarial attacks: 2 white-box attacks (FGSM and BIM), 1 black-box attack (one-pixel attack), and 1 proposed black-box attack with my ideas. Two white-box attacks perform the same outcome in two different datasets (MNIST and CIFAR-10), which the BIM has a better attack success rate than FGSM's with the same model and the same dataset. However, there is still an issue, that is no free lunch theory. BIM spent much more time on attacking model than FGSM did, which is reasonable since more iteration steps among the BIM. These two white-box attacks belong to gradient-base type, which means the epsilon value directly influence the output. When the epsilon value increased, the attack performance was improved. However, high epsilon value also makes perturbations become more easily perceptible with human eyes. Thus, for these two white-box attacks, choosing an appropriate epsilon value is the chief factor to deal with the attack issues in the real world.

As for the ideas to improve original one-pixel attack, it is inspired by the basic algorithm of differential evolution. From Su et al. [20] proposed idea, the initial candidate population, which is 400, is kind of large for selecting exact one pixel to perturbated. Since differential evolution lean to find out the probability of successful attacks, in my opinion, it is unnecessary to find the best one to attack. We only need to find a decent one, which can fool the classifier of the model is good enough. As for the decision criteria of the fitness function of the attack, the possibility under 5% is too strict. In this case, it will take too much time to search the best solution with higher price to pay. The reason I set an early-stop criterion before the max iteration is I found some images are just too difficult to attack, while the machine keeps running with a quite low probability to achieve the successful attack if without the early-stop restriction.

Back to the results of my proposed one-pixel attack. By decreasing the candidates' number only, the time of attack was significant reduced while the attack success rate of

the it has only slightly dropped for VGG16 model. On the other hand, increasing the candidates' number will not increase the success rate but spent much more time than implementing the original attack. Then, enlarge the fitness decision criteria only had the same effect, but would take longer attack time. In this case, the size of initializing candidates is a better way to achieve time-efficiency with a decent success rate. As for changing both candidates' population and decision criteria, the effect of reducing attack time is more significant, however, the attack success rate is not decreased too much when compared to the previous experiments. One highlight is the success rate for attacking Resnet18 model is unchanged. This might because of the differences of structures between two models. In this case, my proposed idea can still result in a decent outcome with time-saving feature.

- **Future works**

One future work is to test my proposed work with targeted attacks. All the experiments I have done are only focusing on the untargeted attack. In some way, targeted attacks will be much difficult since we not only want to fool the classifier to make mistakes, but also make it classify the input image to the specific class.

Another potential work is to increase the test sample size. From my project, I only choose 30 and 50 samples to test my proposed work, which is kind of too small when facing the real-world situation. Since the total size of CIFAR-10 is 60,000, 6000 images of each class, it is necessary to experiment the whole test set to further verify my proposed ideas. Also, try other popular dataset, such as ImageNet, is an approachable future work.

Apart from the differential evolution algorithm for one-pixel attack, according to Su et al. [20], some other evolutionary strategies like adaptive differential evolution or covariance matrix adaptation evolution strategy may also improve the performance of current method.

Furthermore, attacking only one pixel is an excessive condition. Changing the situation of changing only one pixel value to several pixels', or a 2X2, 3X3 block perturbation is another possible work.

Bibliography

[1] E. Alpaydin. Introduction to Machine Learning, Third edition. Ed. Cambride, Massachusetts: The MIT Press, 2014.

[2] S. Ben-Yacoub, B. Fasel, and J. Luettin, "Fast face detection using MLP and FFT," in Proc. Second International Conference on Audio and Video-based Biometric Person Authentication (AVBPA'99), 1999, no. CONF, pp. 31-36.

[3] C. M. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.

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

[5] P. P. K. Chan, J. Zheng, H. Liu, E. C. C. Tsang, and D. S. Yeung, "Robustness analysis of classical and fuzzy decision trees under adversarial evasion attack," Applied soft computing, vol. 107, p. 107311, 2021.

[6] P. Civicioglu and E. Besdok. A conceptual comparison of the cuckoosearch, particle swarm optimization, differential evolution and artificial bee colony algorithms. Artificial intelligence review, pp.1–32, 2013.

[7] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. IEEE transactions on evolutionary computation, 15(1): pp.4–31, 2011.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," 2014.

[9] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in 2018
IEEE Symposium on Security and Privacy (SP), 2018, pp. 19-35: IEEE.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, Vol. 220, No. 4598, 1983.

[11] S. Kotyan and D. V. Vargas, “Adversarial Robustness Assessment: Why both L_0 and L_∞ Attacks Are Necessary.” arXiv, Jul. 16, 2020. Accessed: Oct. 08, 2022. [Online]. Available: http://arxiv.org/abs/1906.06026

[12] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale," 2017.

[13] S. Lal, S. U. Rehman, J. H. Shah, T. Meraj, H. T. Rauf, R. Damaševičius, N. A. Mohammed, and K. H. Abdulkareem, "Adversarial Attack and Defence through Adversarial Training and Feature Fusion for Diabetic Retinopathy Recognition, " 2021.

[14] Y. Ming, C. Du, and C. T. Lin, "Semantic Autoencoder and Its Potential Usage for Adversarial Attack," 2022.

[15] U. Ozbulak, M. Gasparyan, W. De Neve, A. Van Messem, "Perturbation analysis of gradient-based adversarial attacks, " 2020.

[16] M. Said, B. M. S. B. Talukder, K. Mishty, and M. T. Rahman, "Attacking Deep Learning AI Hardware with Universal Adversarial Perturbation," 2021.

[17] Y. Senzaki, S. Ohata, and K. Matsuura, "Simple Black-Box Adversarial Examples Generation with Very Few Queries," IEICE Trans. Inf. & Syst., vol. E103.D, no. 2, pp. 212-221, 2020

[18] M. Sharif, L. Bauer, and M. K. Reiter, "On the Suitability of L_p-norms for Creating and Preventing Adversarial Examples," 2018.

[19] D. Stutz, M. Hein, and B. Schiele, “Disentangling Adversarial Robustness and Generalization,” 2018.

[20] J. Su, D. V. Vargas, and S. Kouichi, “One Pixel Attack for Fooling Deep Neural Networks,” *IEEE Trans. Evol. Computat.*, vol. 23, no. 5, pp. 828–841, Oct. 2019, doi: 10.1109/TEVC.2019.2890858.

[21] L. Sun, M. Tan, and Z. Zhou, "A survey of practical adversarial example attacks," Cybersecurity, 2018, Vol.1 (1), p.1-9.

[22] K. Xu, S. Liu, P. Zhao, P.-Y. Chen, H. Zhang, Q. Fan, D. Erdogmus, Y. Wang, and X. Lin. Structured adversarial attack: Towards general implementation and better interpretability. arXiv preprint arXiv:1808.01664, 2018.

[23] X. Yang, H. Li, Y. Yu, X. Luo, T. Huang, and X. Yang, “Automatic Pixel-Level Crack Detection and Measurement Using Fully Convolutional Network: Pixel-level crack detection and measurement using FCN,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 12, pp. 1090–1109, Dec. 2018, doi: 10.1111/mice.12412.

[24] Z. Yuan, J. Zhang, and S. Shan, “Adaptive Perturbation for Adversarial Attack,” arXiv, arXiv:2111.13841, Nov. 2021. Accessed: Aug. 16, 2022. [Online]. Available: http://arxiv.org/abs/2111.13841

[25] F. Zhang, P. P. K. Chan, B. Biggio, D. S. Yeung, and F. Roli, "Adversarial Feature Selection Against Evasion Attacks," IEEE transacation on cybernetics, 2016, Vol.46(3), p.766-777.

[26] T. Zhou, S. Agrawal, and P. Manocha, “Optimizing One-pixel Black-box Adversarial Attacks.” arXiv, Apr. 30, 2022. Accessed: Oct. 07, 2022. [Online]. Available: http://arxiv.org/abs/2205.02116