Generating Targeted Adversarial Attacks and Assessing their Effectiveness in Fooling Deep Neural Networks

by

GAJJAR SHIVANGI BHARATBHAI
202011023

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

June, 2022

Declaration

I hereby declare that

i) the thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

ii) due acknowledgment has been made in the text to all the reference material used.

[illegible]

Gajjar Shivangi Bharatbhai

Certificate

This is to certify that the thesis work entitled Targeted Adversarial Attack Generation has been carried out by Gajjar Shivangi Bharatbhai for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my/our supervision.

[illegible]

Prof. Shruti Bhilare
Thesis Supervisor

[illegible]

Prof. Avik Hati
Thesis Co-Supervisor

Acknowledgments

The M.Tech Thesis as a whole is a worth accomplishing task. Hence, I take this opportunity to express my earnest gratitude to all those who assisted in various capacities in undertaking the thesis work and devising fruitful methodologies.

To start with, I am privileged to express my sense of respect and regard to Prof. Shruti Bhilare for supervising me throughout the work by providing a systematic pathway and help me explore for various possibilites till the end. In addition to that, I thank my thesis co-supervsor Prof. Avik Hati for guiding me to develop effective algorithms by discussing the issues in methods and brainstorming solutions to their core. Along with that, I express my gratitude towards Prof. Srimanta Mandal as well for providing additional guidance in various topics of the thesis.

Along with technical support, mental support is also of utmost necessity. I am thankful to my parents and brother for all the tranquil discussions. Additionally, I am very much grateful to my friends Meet, Darshil, Dhyanil and Tarang for giving immense support in overcoming difficulties faced in the journey of entire thesis work by exchanging their knowledge in various domains.

Any work is impossible without the resources required to complete the tasks. Last but not least, I am grateful to all the DA-IICT support staff for providing the facilites from laboratories to the systems and internet facility.

Contents

Abstract

Deep neural network (DNN) models have gained popularity for most image classification problems. However, DNNs also have numerous vulnerable areas. These vulnerabilities can be exploited by an adversary to execute a successful adversarial attack, which is an algorithm to generate perturbed inputs that can fool a well-trained DNN. Among various existing adversarial attacks, DeepFool, a white-box untargeted attack is considered as one of the most reliable algorithms to compute adversarial perturbations. However, in some scenarios such as person recognition, adversary might want to carry out a targeted attack such that the input gets misclassified in a specific target class. Moreover, studies show that defense against a targeted attack is tougher than an untargeted one. Hence, generating a targeted adversarial example is desirable from an attacker's perspective. In this thesis, we propose 'Targeted DeepFool', which is based on computing a minimal amount of perturbation required to reach the target hyperplane. The proposed algorithm produces minimal amount of distortion for conventional image datasets: MNIST and CIFAR10. Further, Targeted DeepFool shows excellent performance in terms of adversarial success rate.

List of Tables

List of Figures

CHAPTER 1
Introduction

Achieving good classification accuracy is the foremost reason for the wide usage of DNNs in image classification tasks. In spite of attaining acceptable prediction accuracy, DNNs do have gaps that can be exploited by an attacker to generate adversarial examples. An adversarial example is designed by adding a calculated noise to an original input image such that the perturbed input has imperceptible visual changes in the image, and it gets misclassified by a DNN [24]. An algorithm designed to generate adversarial examples is termed as an *adversarial attack*. Adversarial attacks can be divided into different categories depending upon various conditions. Adversarial attacks and its types are discussed in next segment.

1.1 What is an adversarial attack?

Goodfellow et al. in [8] performed an experiment, which is as shown in the Figure 1.1. It can be seen that, the image is quite visibly recognizable to human eye as an image of a panda. When this image was given to a well trained GoogleNet, it gave the prediction as panda with a confidence of 57.5%. However, after adding a well-calculated noise to the image, the model gave a prediction to be a gibbon with even higher confidence of 99.3%. It can be observed that the noise added to the image is imperceptible to the human eye, but for network it is of high significance. The algorithm used to compute such noise, which when added to an image, can fool a well-trained DNN is called an *adversarial attack*. In this particular experiment, the authors had used Fast Gradient Sign Method(FGSM)[8] adversarial attack to compute the noise or perturbation. More about FGSM attack is discussed in Chapter 2. A perturbed input is known as an *adversarial example*.

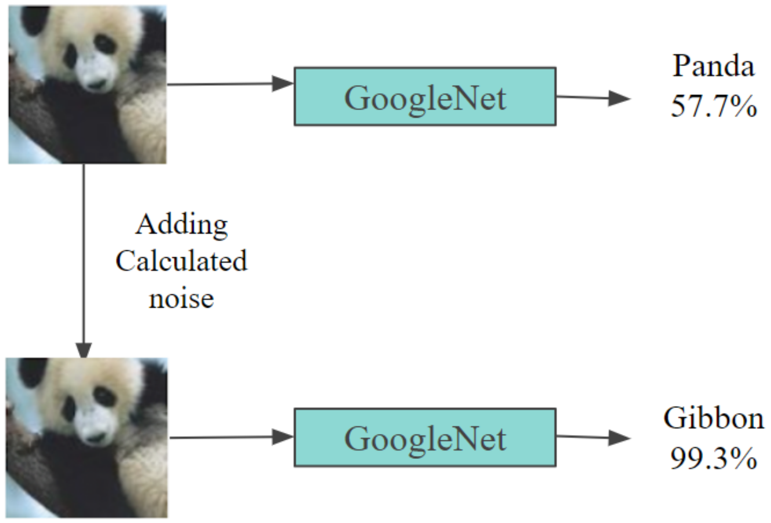


Figure 1.1: Based on an experiment performed in [8]

1.2 Types of adversarial attack

Adversarial attacks can be divided into different types based on various conditions. Based on requirement of architectural knowledge, they are divided into two: white box attack and black box attack. An adversarial attack algorithm that requires architecture knowledge to craft an adversarial example is known as a *white box attack* [28]. The algorithms which do not require architecture knowledge are called the *black box attacks* [28]. Based on constraint of misclassification, adversarial attacks are classified into two: targeted attack and untargeted attack. If an attack algorithm follows a constraint of misclassifying the input into a particular target class, then they are known as *targeted attacks* [20]. The algorithms, which focuses only on misclassifying the input, regardless of any particular target class, are known as *untargeted attacks* [20]. From the mentioned categories of attacks, black-box and targeted attacks are of greater interest in recent research works, as they are difficult to defend.

1.3 Motivation

A question might arise: why to develop an adversarial attack? The answer is simple. In order to find a solution, it is required to study the issue. Similarly, to devise a strong defense, it is required to craft an even stronger attack. As discussed earlier, there exists several works on untargeted attacks. A white-box attack DeepFool [16] is considered one of the most effective untargeted attacks. In general, it is desirable to generate minimum amount of perturbation, and the authors of

DeepFool claim to produce lesser amount of perturbation than the Fast Gradient Sign Method (FGSM) [8]. However, with an increasing amount of research on adversarial defense, simply misclassifying the input is not enough. Therefore, the focus has shifted recently towards targeted attacks, which are difficult to defend than the untargeted ones. Hence, several targeted variants have been proposed in literature for almost all state-of-the-art adversarial attacks. Unlike other algorithms such as FGSM, DeepFool has fewer variants.

1.4 Problem Statement

To achieve targeted misclassification, by proposing a targeted variation of the DeepFool algorithm in the white-box setting, with an aim to study robustness of DNNs against such attacks. We refer to our proposed variant of DeepFool as **Targeted DeepFool**.

1.5 Contribution

The key contributions of the proposed work are as follows:

- To achieve the goal of moving the input towards the target hyperplane using the shortest distance possible, we propose a basic Targeted DeepFool and a recursive Targeted DeepFool. To the best of our knowledge, this is the first such attempt.
- This thesis work discusses the mathematical properties as well as experimental benchmarks set by Targeted DeepFool, in terms of adversarial success rate, while retaining the properties of the original algorithm: minimal amount of perturbation for high natural error.
- In addition to that, average distortion is defined to study the amount of perturbation generated by the algorithm.

1.6 Thesis Outline

The rest of the document is organised as follows. Chapter 2 discusses the previous works on adversarial attacks. Chapter 3 discusses about the original DeepFool algorithm and some analysis on it. Chapter 4 describes the proposed algorithm. Chapter 5 demonstrates the experimental setup for comparing and contrasting the proposed algorithm with other state-of-the-art algorithms. We conclude in Chapter 6.

CHAPTER 2
Literature Review

Szegedy et al. [22] made a breakthrough in the adversarial machine learning area by showing the vulnerabilities of various state-of-the-art DNNs. DNNs trained with high accuracy also have a certain blind spot, where they can be vulnerable to adversarial attacks. These areas are exploited by the adversary to generate adversarial samples. Even by changing the distribution of the input data slightly, a machine learning model can easily be fooled [8]. This property of DNN is used in many adversarial attack algorithms to generate adversarial samples, some of which are discussed in the following segment.

2.1 Fast Gradient Sign Method

Fast gradient sign method (FGSM) is a white-box untargeted attack, which was first introduced by Goodfellow et al. [8]. It is a single step algorithm to generate an adversarial sample x_{adv}, given an original input image x with ground-truth label y_{true} as shown below,

$$x_{adv} = x + \epsilon \cdot \text{sign}(\nabla_x L(h(x), y_{true}))$$

Here, ϵ is a hyperparameter used to tune the amount of perturbation. The intuition behind the algorithm is to maximize the loss ($L(h(x), y)$) of model h between the prediction $h(x)$ of input x with its ground-truth label. Figure 2.1 shows an adversarial image generated using FGSM. Xu et al. [26] proposed a targeted version of FGSM attack in which, the FGSM algorithm is terminated if the image is classified into the target. Iterative FGSM (I-FGSM) is a variant of FGSM, which updates the original input with perturbations computed iteratively using the FGSM single step rule. Momentum iterative FGSM (MI-FGSM) [5] accumulates gradients of the loss function at each iteration in order to escape from poor local maxima. Hence, a stabilize optimization will be achieved. Diverse inputs FGSM (DI-FGSM) [25]

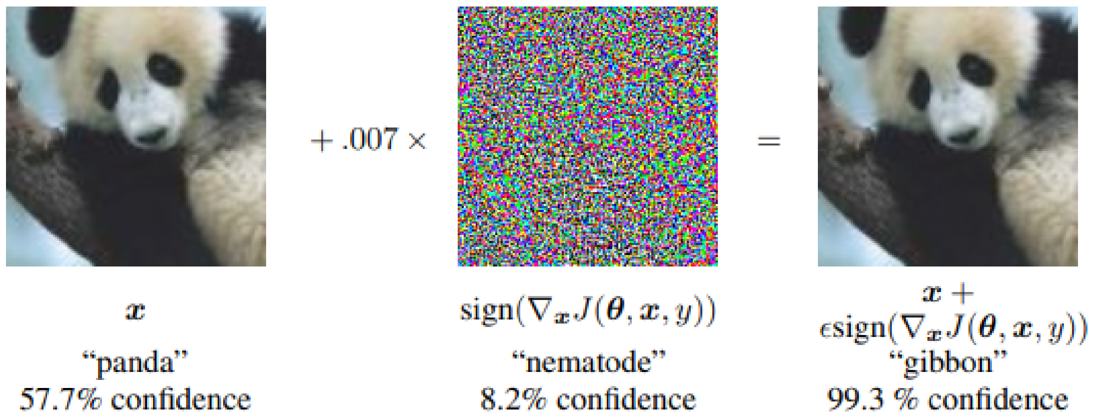

$\boldsymbol{x}$
"panda"
57.7% confidence
$+ .007 \times$
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$=$
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 2.1: An adversarial sample generated using FGSM with $\epsilon = 0.007$[8]

transforms images with a probability to overcome the problem of overfitting of MI-FGSM. In translation invariant FGSM (TI-FGSM) [6], the gradients of the untranslated images are convolved with a predefined kernel, and hence optimize the perturbations.

2.2 Jacobian Saliency Map Attack

Jacobian saliency map attack (JSMA) is a white-box targeted attack, which was first introduced by Papernot et al. [20]. It is a class of algorithms that exploits forward derivatives. The forward derivatives inform the learnt behavior of DNNs and builds an adversarial saliency map to explore efficient adversarial space. Some of the results of JSMA source-target generated examples are shown in Figure 5.4. Jacobian vector gives us the information about the region, where we can most likely craft an adversarial sample to fool the network. Jacobian for a model F given an input X is defined as,

$$J_F(X) = \left[\frac{\partial F(X)}{\partial x_1}, \frac{\partial F(X)}{\partial x_2}\right]$$

where x_1 and x_2 are input neurons. Adversarial saliency maps are defined using Jacobian as,

$$S(X,t)[i] = \begin{cases} 0 \text{ if } J_{ij}(X) < 0 \text{ or } \sum_{j \neq t} J_{it}(X) > 0 \\ J_{it}(X) |\sum_{j \neq t} J_{ij}(X)| \text{ otherwise} \end{cases}$$

These saliency maps give location of the pixels which contributes highly in the classifiation task, which are later used for perturbing the original input. The perturbed images using JSMA on MNIST is shown in Figure 5.4. Greedy JSMA [17]

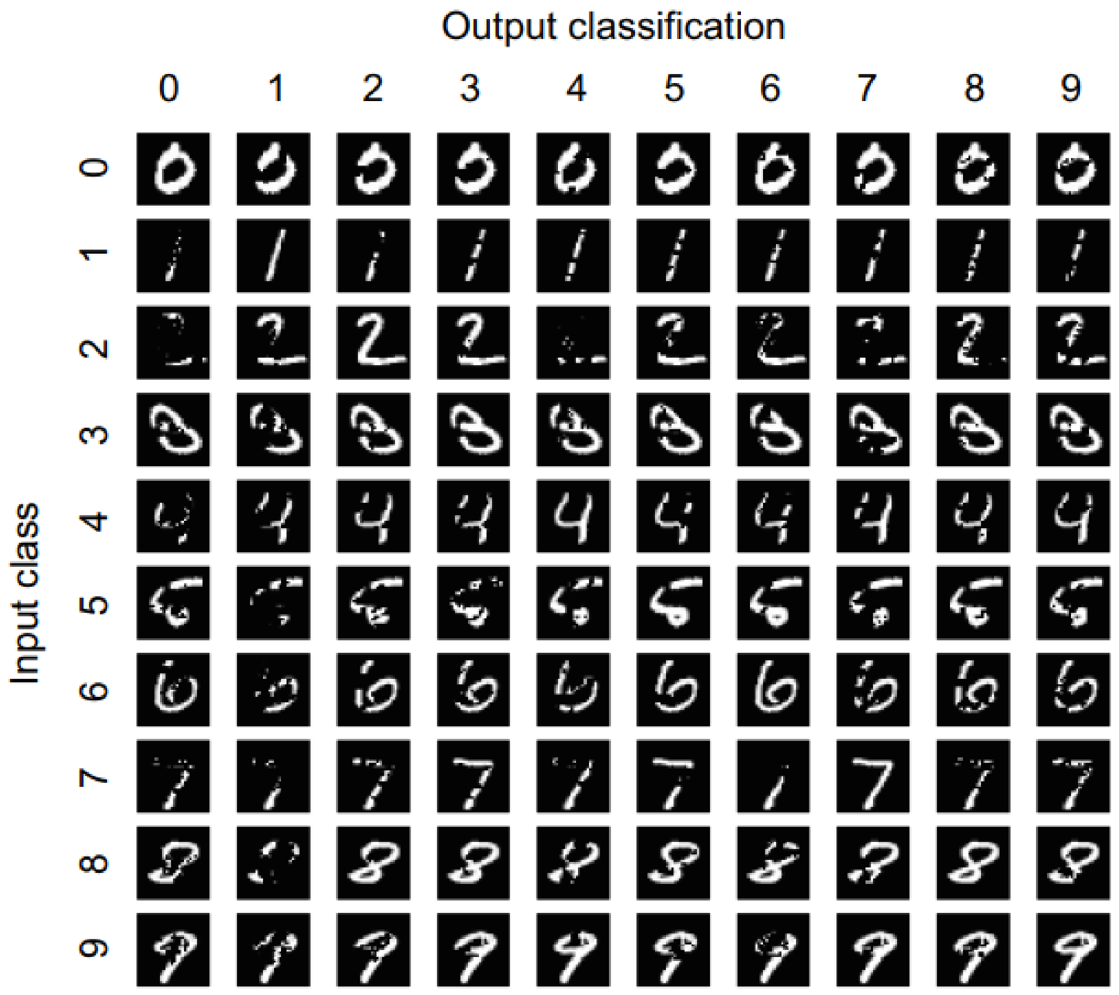


Figure 2.2: Perturbed images generated using JSMA for corresponding source target pairs[20]

is a variation of JSMA. It is a score-based attack. Instead of using Jacobian of network (architecture knowledge), this method performs a greedy local search in the pixel neighbourhood. It is an iterative search process to perturb pixels that have high confidence scores for ground truth.

2.3 Carlini and Wagner Attack

Carlini and Wagner attack (C&W) [2] attack was introduced with a motive of increasing the robustness of DNNs by increasing transferability of adversarial attacks. These attacks are generated using three distance metrics (L_0, L_2, L_∞). It is a black-box targeted attack. In FGSM, any standard loss function like crossentropy is used. However in C&W, the best suited one is the L_2 loss, whose objective function is defined as,

$$\min_w ||\frac{1}{2}(\tanh(w)+1) - x||_2^2 + c \cdot f(\frac{1}{2}(\tanh(w)+1))$$

such that target class $t \neq$ ground-truth and f is given by,

$$f(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa)$$

where Z is the pre-softmax layer and κ is a tuning parameter. Perturbed inputs generated by L_2 C&W on MNIST [4] data are shown in Figure 5.5.

Zeroth order optimization based black-box attack (ZOO) [3] is a variation of C&W score-based attack. It uses zeroth order stochastic co-ordinate descent to directly estimate the gradients of target model to generate adversarial examples.

2.4 DeepFool Attack

Dezfooli et al. [16] introduced the DeepFool algorithm to generate adversarial samples with a motive of adding minimal amount of perturbations to the original image. DeepFool is a white-box untargeted attack. The main idea behind the algorithm is to add perturbation to the input image such that it reaches the other side of the separating hyperplane. This leads to missclassfication of the input. Chapter 3 deals with in-depth discussion of DeepFool algorithm. Figure 5.6 shows a perturbed image generated using DeepFool algorithm compared with perturbations generated using FGSM. The amount of perturbation generated by DeepFool is visibly lesser than that generated by FGSM. All the attacks mentioned till this point are summed up by Goel et al. in [7]. The authors have provided

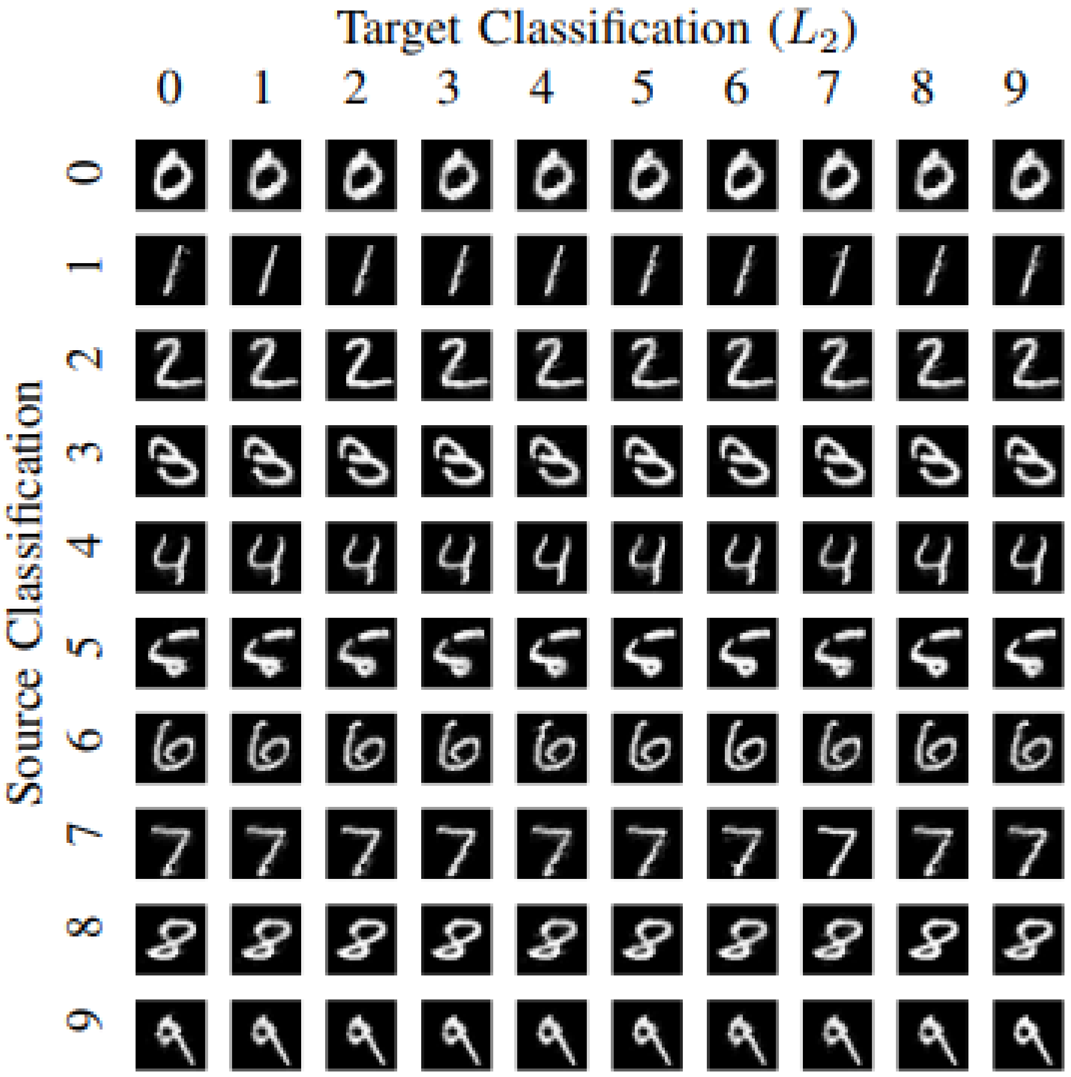


Figure 2.3: Perturbed images generated using C&W L_2 attack for corresponding source-target pairs [2]

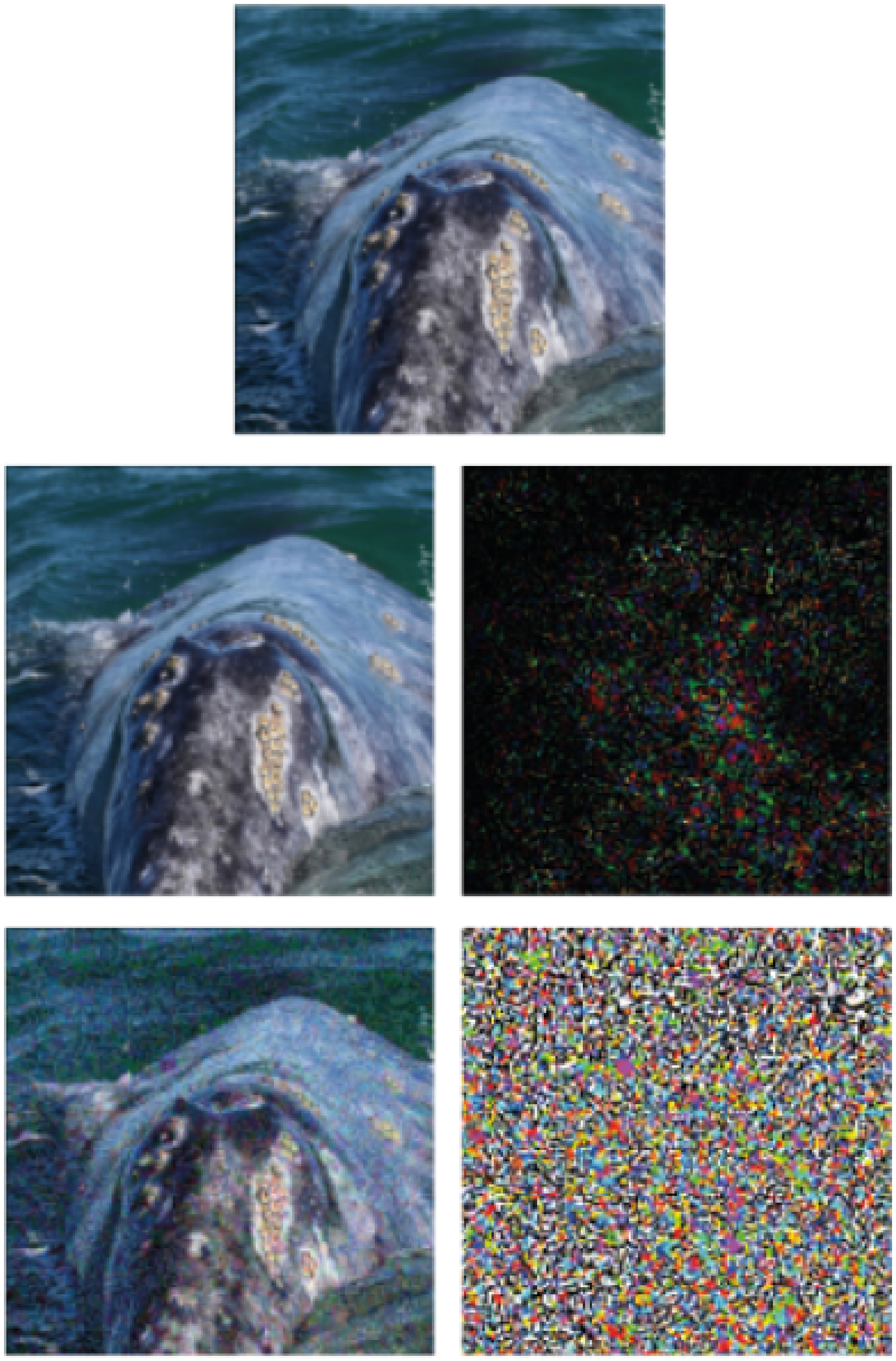


Figure 2.4: Figure taken from DeepFool original paper [16] - Row 1: Original image, Row 2: Perturbed image and perturbations using DeepFool, Row 3: Perturbed image and perturbations using FGSM [8]

a tool called SmartBox, where they have incorporated all the state-of-the-art attacks, with a motive of benchmarking the performance of the mentioned attack algorithms. Along with that, they have also proposed a method of adversarial attack by adding Gaussian noise to the image. However, the experiments were satisfactory as compared to other state-of-the-art attacks.

2.5 Other Attacks

As discussed earlier, there are different categories of attacks based on different scenarios. Apart from targeted-untargeted and black-box - white-box attacks, there is one more category of attack called the universal attack. Universal mapping to generate adversarial examples that can fool classifier to classify all of them to one targeted class and also having strong transferability is a type of attack which is generated in most harsh conditions [24]. Wu in [24] has described a fooling transfer net (FTN), which generates universal perturbations using the low frequency fooling images. Figure 2.5 shows the diagram of FTN (encoder-decoder model). It uses Adaptive Instance Normalization(AdaIN) residual blocks, which are efficient in transfer learning due to the property of real-noise denoising[10]. First the low frequency fooling image is generated for the target label and given to the classifier network. Techniques like transformation robustness(TR), Decorrelation(DR) [19], Compositional Pattern Producing Network (CPPN) [18, 21] and Gradient CPPN are used for generating the low-frequency fooling images. The mean and variance of each layer are stored and given to AdaIN decoder of FTN. When the source image(I_s) is given to FTN, it produces a desired adversarial image (I_a).

Many of the machine learning models give correct output even in a setup of adversarial inputs [8, 14, 27]. Hence, with the motivation to develop a stronger attack, Laidlaw et al. [12] show one more category of attack known as the targeted adversarial attack with an abstain option. The main intuition is to abstain from assigning any class, when the input is not misclassified correctly into the target class. The motive behind abstaining is to reduce the adversarial error, which is considered as a primary metric in comparing targeted adversarial attacks.

Li et al. [13], claim that there are two issues in generating transferable targeted adversarial examples: First, noise curing produced by iterative attacks and second, just getting close to target class without moving away from original class is not enough. Noise curing is a problem with many iterative attacks, like MI-FGSM [5]. Assume a scenario where he magnitude of gradient continuously decreases and at one point vanishes away. However, MI-FGSM rescales it to unit L_1 ball

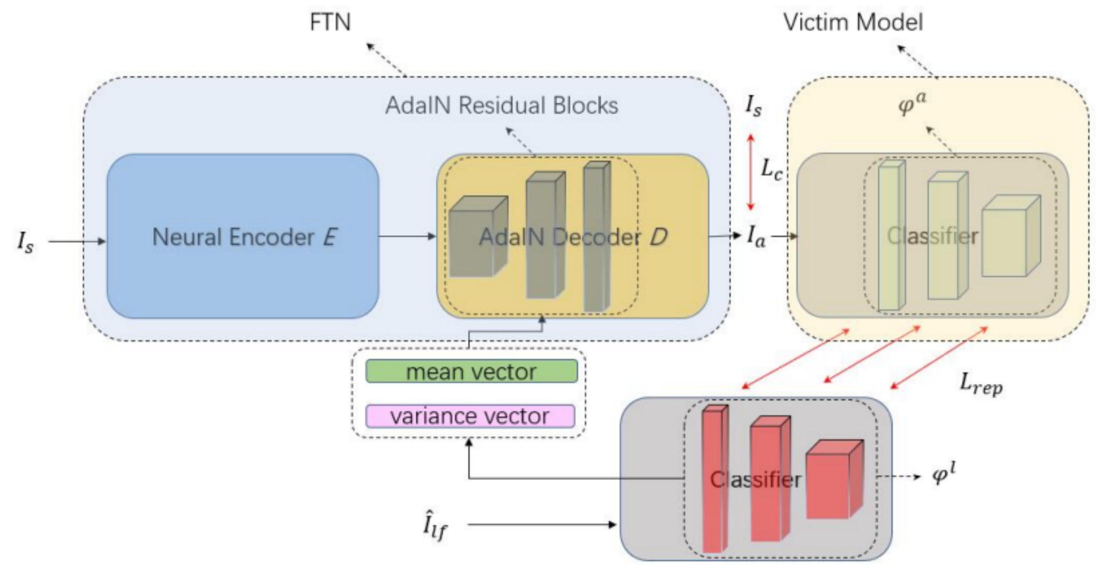


Figure 2.5: Fooling Transfer Net (FTN) [24]

so it doesn't vanish away. But after few iterations the direction of gradient becomes constant showing no bigger effect of addition of noise. This phenomenon is termed as noise curing. To overcome the mentioned two issues, Li et al. [13] proposed an effective approach to generate such examples using Poincarè metric learning (to alleviate noise curing) along with triplet loss to increase the transferability. For any two points u and v in L_1 ball, Poincarè distance is defined as,

$$d(u,v) = \operatorname{arccosh}(1 + \delta(u,v))$$

where

$$\delta(u,v) = 2\frac{||u-v||_2^2}{(1-||u||_2^2)(1-||v||_2^2)}$$

The triplet loss function to be minimized for input x_i with ground-truth y_{true}, to misclassify it into target class y_{tar} is defined as,

$$L_{trip}(y_{tar}, l(x_i), y_{true}) = [d(l(x_i), y_{tar}) - d(l(x_i), y_{true}) + \gamma]$$

where fused logits for K classifiers is given by,

$$l(x) = \sum_{k=1}^{K} w_k l_k(x)$$

Some results are shown in Figure 2.6. The concept used in the Poincarè metric learning is based on the distance measures used in the hyperbolic space. Poincarè

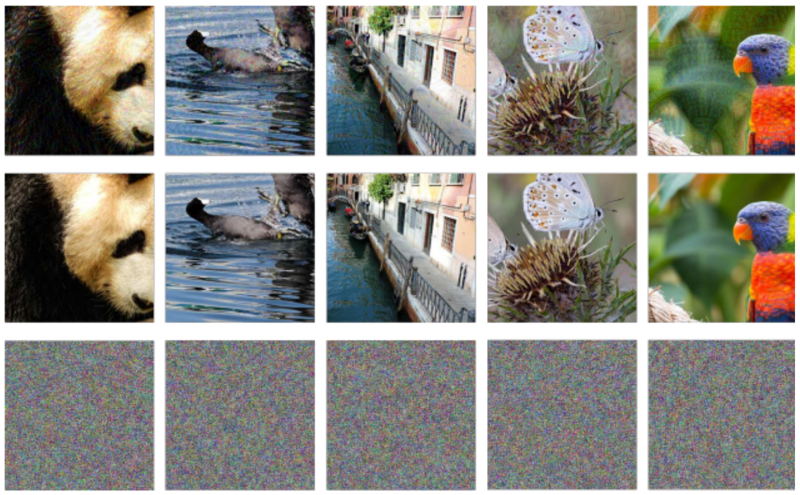


Figure 2.6: Row 1: Original images, Row 2: Perturbed images generated by Poincarè metric learning and Row 3: Corresponding perturbations [13]

ball is a representation of hyperbolic space which has been of greater interest in recent reasearches, as it can efficiently deal with problems having complex data distributions [1, 15].

Apart from these, there are numerous other variations of various state-of-the-art adversarial attacks. Though being a state-of-the-art attack, there are not many targeted variations of DeepFool [16] attack. DeepFool also has some limitations. Along with this, converting an untargeted attack into a targeted one comes with certain challenges, which is discussed in the next chapter with mathematical arguments.

Chapter 3
The DeepFool Attack

3.1 Intuition

The simple fundamental followed in generating adversarial images using DeepFool is to make the input to cross the closest decision boundary in a multiclass scenario. More formally, the input is taken towards the closest hyperplane, in the opposite direction of its gradient. For a binary linear classifier, the equation of perturbations for first iteration is given by,

$$\begin{aligned} p_*(x_0) &= \arg\min_{\|p\|_2}, sign(h(x_0+p)) \neq sign(h(x_0)) \\ &= -\frac{h(x_0)}{\|g\|_2^2} g \end{aligned} \tag{3.1}$$

where $p_*(x_0)$ represents computed perturbation of input x_0 for a model $h(x)$ and g represents weight vector.

For linear multiclass classifiers, the closest hyperplane is computed by,

$$l(x_0) = \arg \min_{k \neq k'(x_0)} \frac{|h_k(x_0) - h_{k'(x_0)}(x_0)|}{\left\| g_k - g_{k'(x_0)} \right\|_2} \tag{3.2}$$

Minimum perturbation is given by,

$$p_*(x_0) = \frac{|h_{l(x_0)}(x_0) - h_{k'(x_0)}(x_0)|}{\left\| g_{l(x_0)} - g_{k'(x_0)} \right\|_2} (g_{l(x_0)} - g_{k'(x_0)}) \tag{3.3}$$

The algorithm follows an iterative update of the above equation, until misclassification is achieved. In practice, not all classifiers are linear. Hence, the linear approximation of hyperplane is considered instead. In brief, the goal of DeepFool is to take the input away from the original label regardless of any specific target class.

3.2 Gaps in DeepFool

The mentioned equations give two inferences: first, the model's knowledge ($f(x_0)$) is used while computing the perturbations and second, misclassification of class, i.e. the target label, is not taken into consideration. Both these observations conclude that DeepFool is not a targeted attack and has poor transferability. Apart from these straightforward conclusions, few researches have explored other gaps in the DeepFool algorithm as well. It is mentioned in [23], even though DeepFool claims to generate minimum perturbations by stepping the closest boundary, there could exist an alternate region where minimum perturbations can be obtained. Because of this property, DeepFool still needs improvement in untargeted setting as well. One reason behind this, is the gaps or holes produced while computing the linear approximation of decision boundary, which is mathematically proved in [9] and [23]. The algorithm is designed in a way that it needs to use the model parameters to generate the adversarial examples. Studies show that improving transferability is more of a difficult task than to make a targeted version of same.

Chapter 4
Proposed Method: Targeted DeepFool

In order to devise an adversarial attack algorithm, we used the idea of DeepFool[16]. Along with that, we introduced phenomenon of target misclassification. In the process of devising the algorithm for Targeted DeepFool, we first propose a basic algorithm for the same. Basic algorithm encounters an issue of input getting stuck on the intermediate hyperplanes, which is discussed in detail, in the upcoming segment. In order to resolve the issue, we then modified the algorithm by introducing recursion in the algorithm.

4.1 Basic Targeted DeepFool

4.1.1 Intuition

Basic algorithm of Targeted misclassification is based on the fundamental of pushing the input towards the target hyperplane using the concept of orhtogonal projection. Figure 4.1 shows an instance of a non-linear multiclass classifier. Among multiple seperating hyperplanes, the red one (H_4) is the target hyperplane and the rest of the hyperplanes are intermidiate hyperplanes. However, not all the hyperplanes are linear. So, first a linear approximation of the hyperplanes is computed (represented using dotted lines). Our goal is to push the input x towards the target hyperplane H_4. The shortest path to reach the hyperplane is using the orthogonal projection of x on H_5 (red arrow). In DeepFool[16], the orthogonal projection with respect to the closest hyperplane (H_3) is considered (blue arrow). The intuition behind basic algorithm is supported using mathematical arguments and derivation in the next segment.

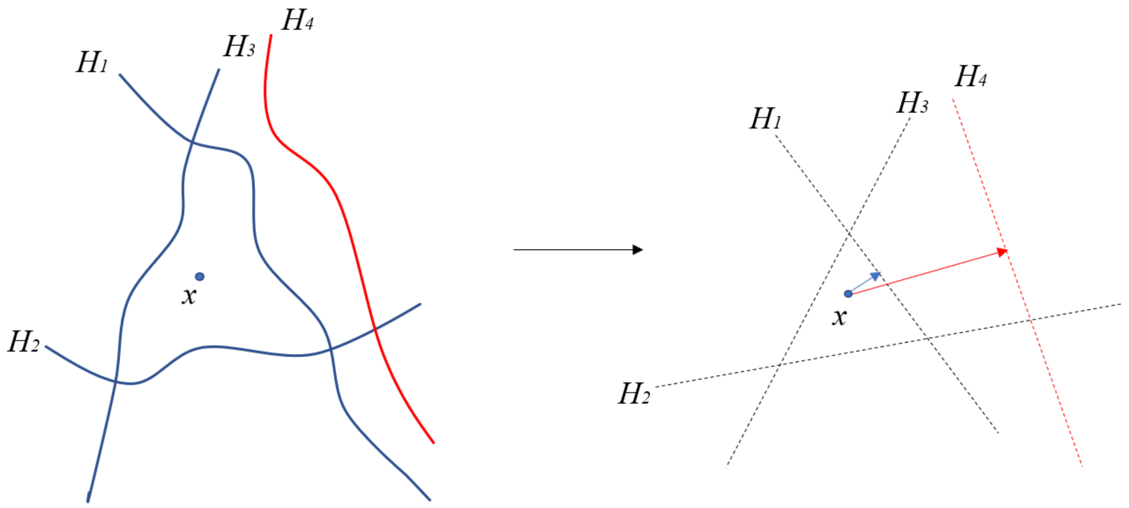


Figure 4.1: Intuition of basic Targeted DeepFool using an instance of mutliclass non-linear classifier approximated to be near to linear.

4.1.2 Mathematical Derivation

Consider a binary classifier $h(x) = g^T x + g_0$ (Fig. 4.2), where g is the learnable weight vector, g_0 is the bias. The hyperplane separating the two classes is defined as $H = \{x : h(x) = 0\}$. Our goal is to craft an adversarial sample (x), by perturbing the input(x_0), such that it reaches the hyperplane H, i.e.

$$h(x) = 0 \tag{4.1}$$

However, in practice, not all classifiers are linear. So, it is required to take an approximation instead. For this, we can use Taylor series expansion of h as:

$$h(x) = h(x_0) + \nabla_x h(x_0)^T (x - x_0), \tag{4.2}$$

considering the higher order terms to be zero.

Now, our optimization problem is to get the minimum amount of perturbation between adversarial input x and original input x_0. However, it is a challenging task to perturb the input using a single step algorithm. Hence, perturbation $p_i = x_{i+1} - x_i$ are computed iteratively to obtain an adversarial example as:

$$\min_{p_i} \|x_{i+1} - x_i\|_2^2, \tag{4.3}$$

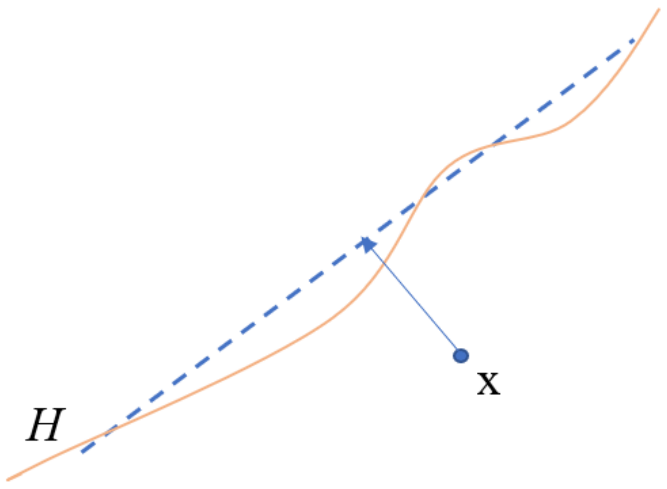


Figure 4.2: Binary classifier

with the goal to reach the hyperplane H, that is,

$$h(x_{i+1}) = h(x_i) + \nabla_x h(x_i)^T (x_{i+1} - x_i) \approx 0. \tag{4.4}$$

Therefore, the overall optimization problem becomes:

$$\begin{aligned} \min_{p_i} \frac{1}{2} \|x_{i+1} - x_i\|_2^2 \\ + \lambda (h(x_i) + \nabla_x h(x_i)^T (x_{i+1} - x_i)) \text{sign}(h(x_i)) \end{aligned} \tag{4.5}$$

It can be shown that $\nabla_x h(x)$ represents the learnable weight vector (g) of the classifier. Using first order optimization condition to solve the constrainted optimization problem, derivative of Eq. (4.5) with respect to λ gives,

$$\begin{aligned} g^T x_{i+1} + g_0 = 0 \\ g^T x_{i+1} = -g_0 \end{aligned} \tag{4.6}$$

Now, differentiating Eq. (4.5) with respect to p_i gives,

$$x_{i+1} - x_i + (\lambda g) sign(h(x_i)) = 0 \tag{4.7}$$

Multiplying Eq. (4.7) by g^T,

$$\begin{aligned} g^T x_{i+1} - g^T x_i + \lambda \|g\|_2^2 sign(h(x_i)) = 0 \\ \lambda = \frac{g^T x_i - g^T x_{i+1}}{\|g\|_2^2} sign(h(x_i)) \end{aligned} \tag{4.8}$$

From Eq. (4.6),

$$\begin{aligned}\lambda &= \frac{g^T x_i + g_0}{\|g\|_2^2} sign(h(x_i)) \\ \lambda &= \frac{h(x_i)}{\|g\|_2^2} sign(h(x_i))\end{aligned} \tag{4.9}$$

Putting value of λ in Eq. (4.7),

$$\begin{aligned}x_{i+1} &= x_i - \frac{h(x_i)}{\|g\|_2^2} g \\ &= x_i - \frac{h(x_i)}{\|\nabla_x h(x_i)\|_2^2} \nabla_x h(x_i) \\ &= x_i + p_i\end{aligned} \tag{4.10}$$

where p_i is the amount of perturbation added to the input iteratively until misclassification is achieved.

Now, considering h to be a multiclass classifier, in order to craft an adversarial sample that gets classified into a target class t from an input with ground-truth label $j(x_i)$, the input is pushed towards the target hyperplane iteratively. Hence, deriving from Eq. (4.10), the perturbation (p_i) are computed as:

$$p_i = \frac{\left| h_t\left(x_i\right) - h_{j(x_i)}\left(x_i\right) \right|}{\left\| g_t - g_{j(x_i)} \right\|_2^2} \left(g_t - g_{j(x_i)} \right), \; x_{i+1} = x_i + p_i \tag{4.11}$$

until the input is classified into the target class t, and $h_t(x)$ is the classifier prediction for class t. The basic algorithm of Targeted DeepFool works on this basic update rule (Eq. (4.11)).

The algorithm is summarized in Algorithm 1 as an iterative process, which takes an input image x, model h and target hyperplane t as input arguments. The adversarial sample is initialized (x_0) with the original input image and gets updated by perturbation p_i obtained in every iteration i. The total perturbation p_{tot} required to generate an adversarial sample from x_0 is computed by summing up the perturbations p_i obtained over all iterations. Here, a minimal amount of perturbation is ensured by computing the orthogonal projection of the image on the target hyperplane. However, there is limited control over the amount of perturbation if the target hyperplane is farther away. Hence, a threshold value is set for the amount of perturbation to ensure that the algorithm terminates when it encounters a non-converging situation.

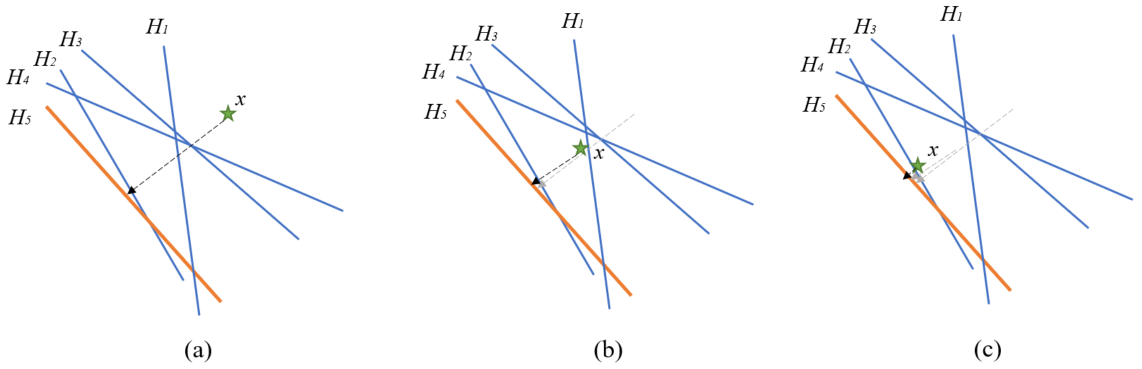


Figure 4.3: (a) Input x is closest to H_1 hyperplane initially, (b) After execution of basic algorithm of Targeted DeepFool it has reached the boundary of H_3, but could not reach H_5 hence recursion will take place, (c) After one recursion, the input is much closer to target hyperplane H_5, which was not possible with basic algorithm.

Algorithm 1 Basic Targeted DeepFool

```
procedure GENERATEADV($x, h, t, p_{tot}$)
    Initialize $x_0 \leftarrow x, i \leftarrow 0$.
    Set *threshold* (maximum perturbation)
    while $\arg\max_j h_j(x_i) \neq t$ or $p_i < threshold$ do
        $p_i \leftarrow \frac{|h_t(x_i) - h_{j(x_i)}(x_i)|}{\left\| g_t - g_{j(x_i)} \right\|_2^2} (g_t - g_{j(x_i)})$
        $x_{i+1} \leftarrow x_i + p_i$
        $i \leftarrow i + 1$
    end while
    $p_{tot} \leftarrow \sum_i p_i$
    return $p_{tot}$
end procedure
```

4.1.3 Issues faced

The main idea behind the basic algorithm is to push the input towards the target hyperplane. However, estimating a hyperplane for a DNN is a non-trivial task, which makes the optimization used in DeepFool intractable [23]. As discussed earlier, one reason behind this is the gaps or holes produced while computing the linear approximation of the decision boundary, which is mathematically proved in [9] and [23]. Basic algorithm of Targeted DeepFool also faces the same issue. Due to inefficient approximation of non-linear hyperplanes, the algorithm might face a scenario where the input gets stuck on an intermediate hyperplane.

Fig. 4.3(a) shows one such scenario. Initially, the example is closest to the hyperplane H_1, and the target hyperplane is H_5. According to the basic algorithm, the orthogonal projection of x on H_5 gives the minimum amount of perturbation to be added to make x reach the boundary of H_5. However, due to high density of other hyperplanes on the way, x gets stuck on the boundary of an intermediate hyperplane. The basic algorithm does not have a provision for retracting and continuing when it gets stuck on an intermediate hyperplane (Fig. 4.3(b)).

4.2 Recursive Targeted DeepFool

4.2.1 Issue Resolved

As discussed earlier, we observed that many of the inputs were getting stuck on an intermediate hyperplane (Fig. 4.3(b)). After that we performed experiments on the input by considering the ground truth as the poisiton where the input got stuck. We observed that, the input was getting even more closer to the target hyperplane (Fig. 4.3(c)). That is how we came up with the modified version of basic algorithm, known as recursive algorithm.

4.2.2 Intuition

Algorithm 2 Recursive Targeted DeepFool

```
procedure GenerateAdv($x$, $h$, $t$, $recLimit$, $p_{tot}$)
    Initialize $x_0 \leftarrow x$, $i \leftarrow 0$, $maxRec \leftarrow 15$.
    Set $threshold$ (maximum perturbation)
    while $\arg\max_j h_j(x_i) \neq t$ or $p_i < threshold$ do
        $p_i \leftarrow \frac{|h_t(x_i) - h_{j(x_i)}(x_i)|}{\left\| g_t - g_{j(x_i)} \right\|_2^2} (g_t - g_{j(x_i)})$
        $x_{i+1} \leftarrow x_i + p_i$
        $i \leftarrow i + 1$
    end while
    $p_{tot} \leftarrow p_{tot} + \sum_i p_i$
    if $\arg\max_j h_j(x_i) \neq t$ and recLimit $< maxRec$ then
        call GenerateAdv($x_i$, $h$, $t$, $recLimit + 1$, $p_{tot}$)
    end if
    return $p_{tot}$
end procedure
```

We propose a recursive version of the basic algorithm that resolves the above mentioned issue to a great extent. As shown in Algorithm 2, if convergence is not

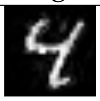
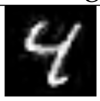
	Basic Algorithm	Recursive Algorithm
Perturbed images		
Source ∣ Target	4 ∣ 5	4 ∣ 5
Iteration (total)	1000*	100
L_2 norm of perturbations	14175.311	13631.604

Table 4.1: Comparison between basic and recursive algorithms Targeted DeepFool for a sample from MNIST dataset with no threshold limit on amount of perturbations. (* = manually stopped)

achieved by the basic algorithm, the perturbed sample generated as its output is used as the input for the subsequent call of the recursive algorithm. The recursions terminate when the target misclassification is achieved. However, it is still possible to encounter a scenario where the generated example jumps between intermediate hyperplanes and never reaches the target hyperplane. For such cases, an upper bound on the number of recursions ($maxRec$) is set as the stopping criterion. $recLimit$ keeps a track of the number of recursions, which is initially set to 0. For experimental purposes, we have set the maximum recursion limit ($maxRec$) to be 15. The total perturbation p_{tot} required to generate the desired adversarial sample is initialized to 0. Then in each recursion, perturbations obtained over all recursions are summed up and used to update p_{tot}.

Table 4.1 shows comparitive analysis between basic algorithm and recursive algorithm without any threshold perturbation limit, for an image of handwritten digit 4. When the input was given to the basic algorithm with target specified as 5, it ran for more than 1000 iterations, where we had to manually terminate the algorithm. On the other hand, recursive algorithm converged after 1 recursion, i.e. 100 iterations in total. The L_2 norm of the perturbations show that the recursive algorithm not only resolves the issue to desirable extent, but also produces lesser amount of perturbations.

4.3 Variation 1: Recursive algorithm with a provision of skipping visited hyperplane

In the process of developing the recursive algorithm for Targeted DeepFool, we observed that the algorithm takes equal time to process as that of the original DeepFool [16]. Hence, to reduce the time we came up with a modified version of Algorithm 2. The intuition behind the variation is to keep a track of visited intermediate hyperplane and skip the hyperplane when the input gets stuck again on the same hyperplane. In this case the only option was to select any random hyperplane from the non-visited list of hyperplane as an initial hyperplane for next recursion. Algorithm 3 shows the modified algorithm. Here, *visited* is initally empty set and *not_visited* is a set of all labels, except the ground-truth label. The modification can be visible at the part, where the condition of whether the label belongs to *visited* set or not, is specified. If the *label* is present in the *visited* set, we select any label from the *not_visited* set. Otherwise, the label is removed from the *visited* set and rest of the algorithm is executed as the recursive algorithm.

However, doing so, the algorithm couldn't fetch desirable results as the input was assumed to be stuck on non-visited hyperplane, but in reality it was not at that hyperplane. Hence, to overcome this issue, we performed one more modification the algorithm, which is discussed in the upcoming section.

Algorithm 3 Recursive Targeted DeepFool : Variation 1

```
procedure GENERATEADV($x$, $h$, $t$, $recLimit$, $p_{tot}$, $visited$, $not\_visited$)
    Initialize $x_0 \leftarrow x$, $i \leftarrow 0$, $maxRec \leftarrow 15$.
    Set $threshold$ (minimum perturbation)
    $label \leftarrow j(x)$
    if $label \in visited$ then
        $label \leftarrow not\_visited$.pop(0)
    else
        Remove $label$ from $not\_visited$
    end if
    Insert $label$ in $visited$
    while $h(x_i) \neq y_t$ or $p_i < threshold$ do
        $p_i \leftarrow \frac{|h_t(x_i) - h_{label}(x_i)|}{\|g_t - g_{label}\|_2^2}(g_t - g_{label})$
        $x_{i+1} \leftarrow x_i + p_i$
        $i \leftarrow i + 1$
    end while
    $p_{tot} \leftarrow p_{tot} + \sum_i p_i$
    if $h(x_i) \neq y_t$ and $recLimit < maxRec$ then
        call GENERATEADV($x_i$, $h$, $t$, $recLimit + 1$, $p_{tot}$, $visited$, $not\_visited$)
    end if
    return $p_{tot}$
end procedure
```

4.4 Variation 2: Adding constraint in the algorithm to skip visited hyperplanes

As mentioned in the modified version of recursive algorithm, the input is not able to reach the target hyperplane by simply skipping the visited hyperplanes. Apparently, there are cases, where the input is not able to easily move away from any particular hyperplane. Due to this, skipping the visited hyperplane, might not add proper amount of perturbation. Therefore, we came up with a modification in first variation, where we allow the hyperplane to be visited for *label_count* times. For experiments we used $label\_count = 3$ are achieved desirable results. However, after all the experiments, we found that the Algorithm 2 is accomplished in itself than the other variations.

Algorithm 4 Recursive Targeted DeepFool: Variation 2

```
procedure GenerateAdv($x$, $h$, $t$, $recLimit$, $p_{tot}$, $visited$, $not\_visited$, $label\_count$)
    Initialize $x_0 \leftarrow x$, $i \leftarrow 0$, $maxRec \leftarrow 15$.
    Set $threshold$ (minimum perturbation)
    $label \leftarrow j(x)$
    if $label \in visited$ && $label\_count < 2$ then
        $label \leftarrow not\_visited$.pop(0)
        $label\_count \leftarrow label\_count + 1$
    else
        Remove $label$ from $not\_visited$
    end if
    Insert $label$ in $visited$
    while $h(x_i) \neq y_t$ or $p_i < threshold$ do
        $p_i \leftarrow \frac{|h_t(x_i) - h_{label}(x_i)|}{\|g_t - g_{label}\|_2^2}(g_t - g_{label})$
        $x_{i+1} \leftarrow x_i + p_i$
        $i \leftarrow i + 1$
    end while
    $p_{tot} \leftarrow p_{tot} + \sum_i p_i$
    if $h(x_i) \neq y_t$ and $recLimit < maxRec$ then
        call GenerateAdv($x_i$, $h$, $t$, $recLimit + 1$, $p_{tot}$, $visited$, $not\_visited$,$label\_count$)
    end if
    return $p_{tot}$
end procedure
```

CHAPTER 5
Experimental Setup

Recursive Targeted DeepFool resolves the issue faced in the basic Targeted DeepFool to a desirable extent. In addition to that, on experimental grounds, the recursive algorithm is found to be more effective than the basic one. Hence, in the rest of the document, 'Targeted DeepFool' refers to the recursive algorithm (Algorithm 2) if not specified otherwise.

5.1 Datasets

Targeted DeepFool is tested on two widely used image datasets MNIST [4] and CIFAR10 [11]. MNIST dataset is a collection of 28×28 MNIST grayscale images of handwritten digits from 0 to 9. It consists of 60,000 images in training set and 10,000 images in validation set. CIFAR10 dataset consists of 32×32 RGB images for 10 classes. It has 50,000 images in training set and 10,000 images in validation set.

5.2 Model Architectures

Targeted DeepFool is a white-box attack. Hence, it requires the knowledge of the architecture and model parameters to generate adversarial samples. Figures 5.1 and 5.2 shows the model definitions for MNIST and CIFAR10, which are identical to that of Carlini and Wagner [2] for generating transferable adversarial examples. MNIST model is trained with a dropout of 0.5 and test accuracy of 99.1% on validation set. CIFAR10 model is also trained with the same dropout with test accuracy of 79% on validation set.

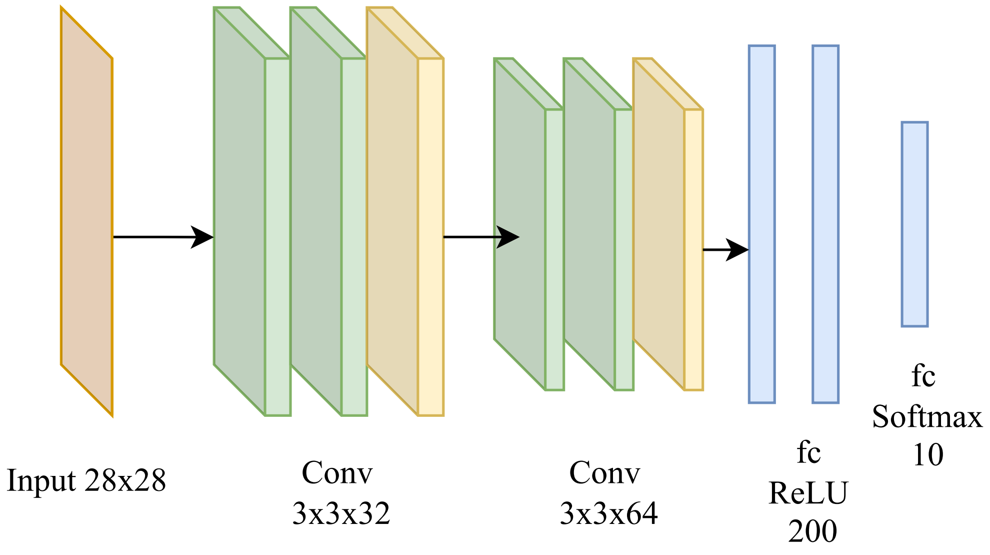


Figure 5.1: Architecture for pretraining the model for MNIST dataset

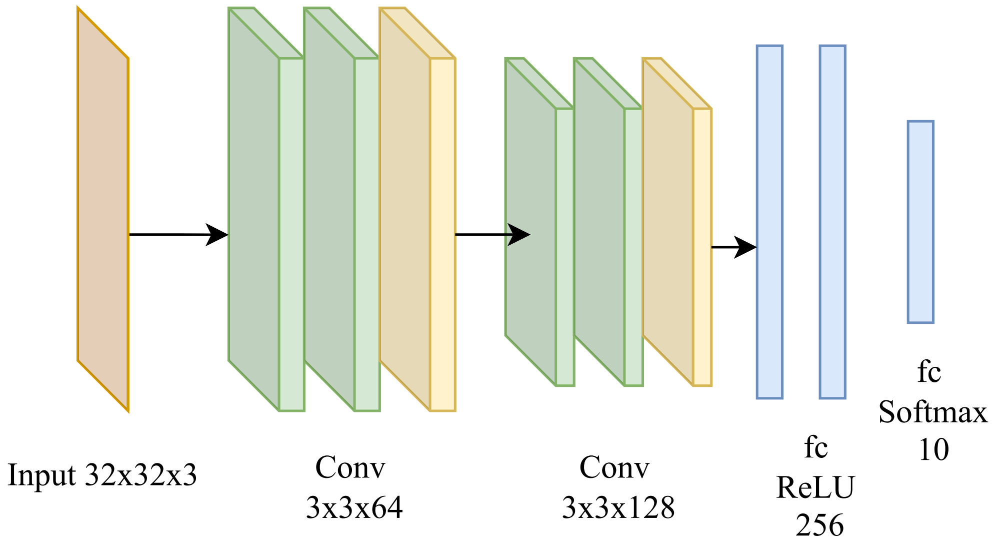


Figure 5.2: Architecture for pretraining the model for CIFAR10 dataset

5.3 Metrics

In order to evaluate the effectiveness of the proposed algorithm, following metrics are used. The percentage of correctly classified images into the target class is known as *adversarial success rate*. The percentage of adversarial images misclassified by the model is known as *natural error*. Additionally, *average distortion* is defined as the fraction of pixels undergoing a pixel value change of more than a certain threshold q.

If N_{total} denotes the total number of pixels, average distortion (δ) is defined as:

$$\delta = \frac{1}{N_{total}} \sum_i 1(|x_{adv}(i) - x_0(i)| > q)\,, \tag{5.1}$$

where x_0 and x_{adv} denote the original and the generated adversarial image, respectively, and $1()$ is an indicator function. Figure 5.3 shows the average distortion for different pixel-difference thresholds. It can be observed that the distortion is decreasing with increase in the pixel-difference. As the average distortion stabilizes when $q > 150$, the value of threshold q is chosen as 150 while reporting the results.

5.4 Results

Table 5.3 shows the adversarial success rate and natural error for MNIST model and CIFAR10 model on 10,000 adversarial samples, generated from the test set of respective datasets. The natural error of the models are much higher, giving an inference that the adversarial attack is successful. However, as the algorithm is a targeted attack, we are more concerned of adversarial success than natural error. MNIST and CIFAR10 models report 97.84% and 77% adversarial success rate, respectively, of recursive Targeted DeepFool. This shows that the algorithm not only generates adversarial samples, but also is able to achieve the goal of crafting targeted adversarial samples.

In Table 5.4, we compare the average distortion of the adversarial samples generated by Targeted DeepFool, JSMA [20], C&W [2], targeted FGSM [26] and Poincarѐ metric learning with triplet loss [13]. It can be observed that Targeted DeepFool yields desirable results as compared to the state-of-the-art targeted attacks in terms of adversarial success rate on MNIST dataset. C&W attack is considered one of the strongest targeted attacks. Despite that, Targeted DeepFool

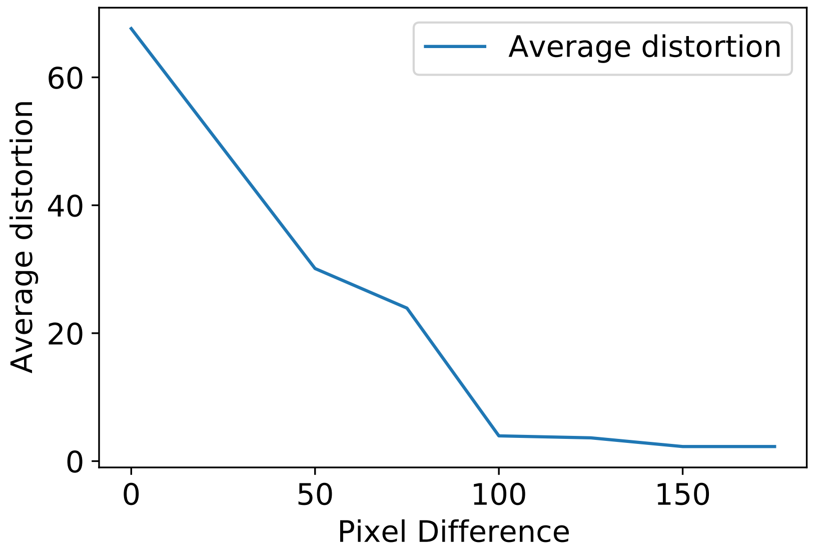


Figure 5.3: Average distortion obtained for adversarial samples generated using Targeted DeepFool algorithm for different pixel-difference thresholds on MNIST dataset.

matches the adversarial success rate of C&W L_2 attack with a close margin. However, in terms of average distortion, Targeted DeepFool yields significantly better results than all attacks including C&W.

Figures 5.4 and 5.5 show the adversarial samples generated by Targeted DeepFool from MNIST and CIFAR10 datasets, respectively. It can be observed that the adversarial images look visually perceptible as the original ones. An algorithm is successful if it works for all types of data. The images shown prove that the algorithm generates less distorted adversarial samples even for RGB images. Tables 5.1 and 5.2 show the L_2 norm of the perturbations added for each adversarial image shown in Figures 5.4 and 5.5. For example, it can be seen from the Table 5.1 that amount of perturbation required to generate an example with ground truth 1 and target 0 is more than that of ground truth 2 and target 0. Hence, these tables show an analysis of amount of perturbation required for an example to convert it into an adversarial example with a particular target class. Figure 5.6 shows visual comparison of images generated by Targeted DeepFool with FGSM [8] attack. A crucial property of DeepFool is that it claims to have lesser perturbation as compared to FGSM [16]. This property is preserved in Targeted DeepFool as well, and Figure 5.6 demonstrates lesser amount of distortion. This observation is supported by the L_2-norm between the original image and the adversarial image reported in Table 5.5.

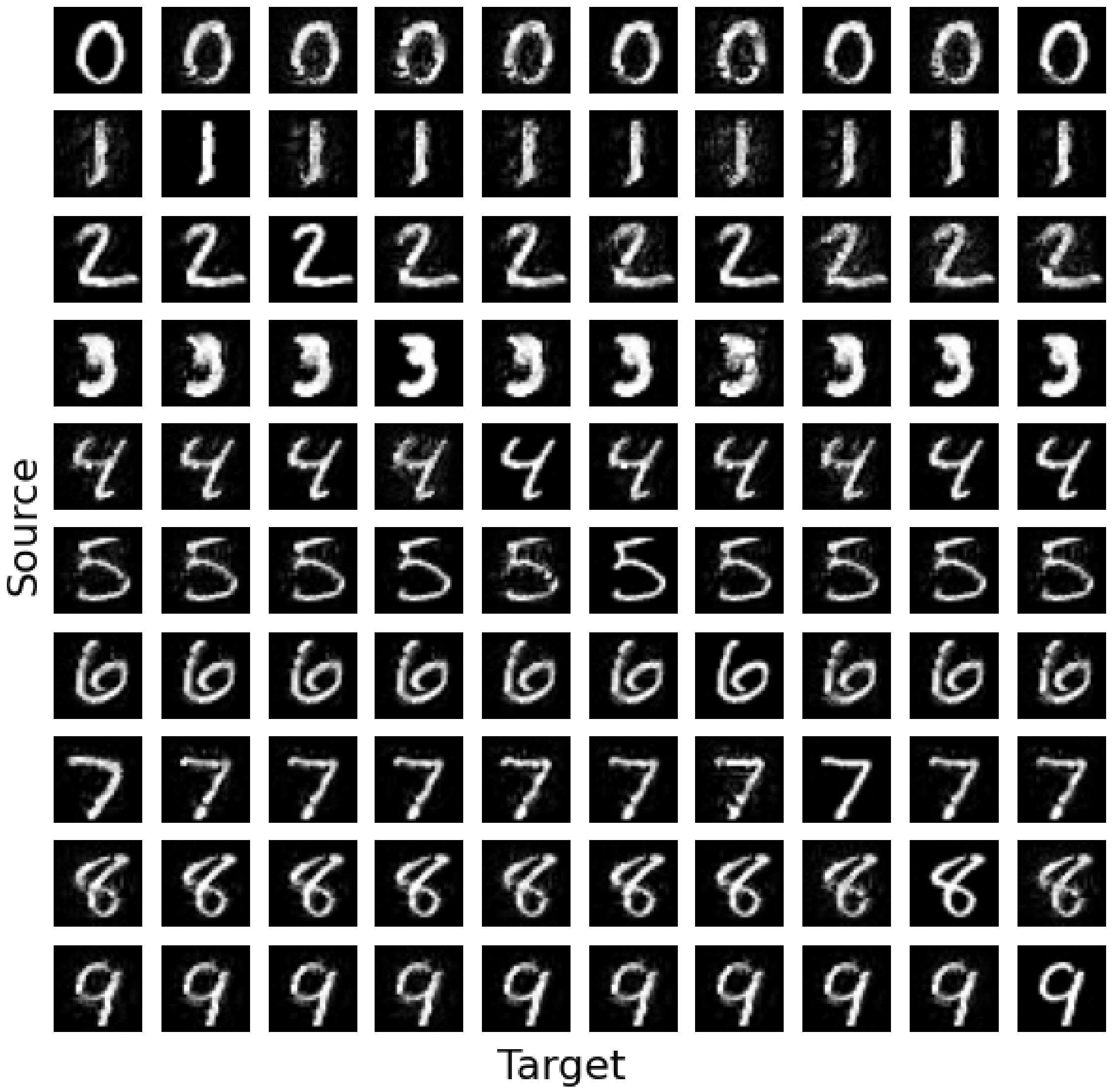


Figure 5.4: Adversarial samples generated by Targeted DeepFool for MNIST dataset.

	0	1	2	3	4	5	6	7	8	9
0	0	9780	10006	9692	9686	8984	9490	8753	9950	7774
1	9900	0	9747	8255	9490	8330	10219	9247	8281	8515
2	7896	8068	0	9172	8722	9783	8065	9708	9948	9804
3	8826	8782	8662	0	8812	7985	9533	8747	7866	7488
4	9230	9288	7945	9940	0	8810	8578	9521	7753	7355
5	9164	8981	8610	7759	9289	0	8688	8757	8572	8701
6	8404	7623	7893	7632	8672	8296	0	9103	7665	8620
7	7929	8314	7919	7953	8904	8010	9338	0	8145	8036
8	9311	8018	8103	8067	9105	7976	8041	9503	0	9656
9	8522	8303	7764	9200	7812	8282	7850	7811	8758	0

Table 5.1: L_2 norm of perturbations for MNIST using Targeted DeepFool

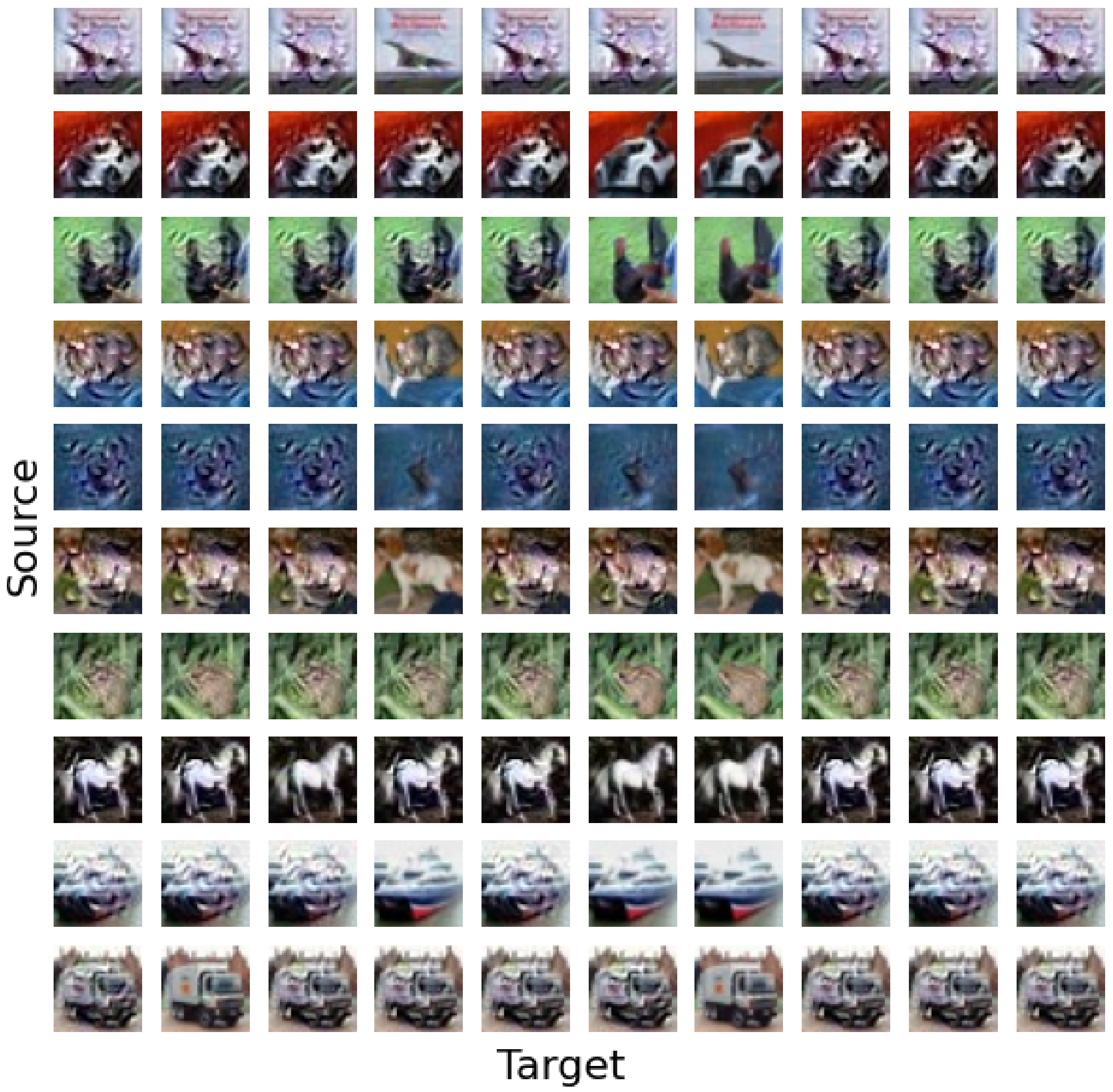


Figure 5.5: Adversarial samples generated by Targeted DeepFool for CIFAR10 dataset.

	0	1	2	3	4	5	6	7	8	9
0	0	9066	9066	9348	9304	9388	9218	9545	9304	9304
1	9304	0	9469	8959	9296	9294	9312	8880	9442	9542
2	9454	9454	0	9384	9405	9043	9105	9442	9542	9231
3	9066	9066	9066	0	9066	9066	9413	9066	9066	9066
4	9545	9545	9545	9304	0	9397	9241	9545	9545	9545
5	9454	9454	9441	9231	9454	0	9223	9454	9454	9454
6	9442	9542	9442	9378	9442	9426	0	9481	9442	9442
7	9469	9469	8959	9296	9674	8880	8983	0	9469	9296
8	9469	8959	9348	9088	8983	9111	9077	9469	0	9348
9	9301	9600	9405	9043	9410	9410	9589	9410	9410	0

Table 5.2: L_2 norm of perturbations for CIFAR10 using Targeted DeepFool

<table>
<tr><th rowspan="2">Dataset</th><th colspan="2">Adv. Success</th><th colspan="2">Natural Error</th></tr>
<tr><th>Basic</th><th>Recursive</th><th>Basic</th><th>Recursive</th></tr>
<tr><td>MNIST</td><td>78.00%</td><td>97.84%</td><td>90.00%</td><td>98.12%</td></tr>
<tr><td>CIFAR10</td><td>65.20%</td><td>77.00%</td><td>78.00%</td><td>89.00%</td></tr>
</table>

Table 5.3: Adversarial success rate and natural error for 10,000 adversarial test samples of MNIST and CIFAR10 generated using basicrecursive Targeted DeepFool.

Attack	Adv. Success	Avg. Distortion
Targeted DeepFool	97.84%	2.28%
C&W L_2 [2]	97.87%	3.10%
JSMA [20]	97.05%	4.45%
Poincare+Triplet [13]	93.97%	11.26%
Targeted FGSM [26]	94.08%	24.31%

Table 5.4: Adversarial success rate and average distortion of 10,000 adversarial samples generated by Targeted DeepFool and other attacks on MNIST dataset.

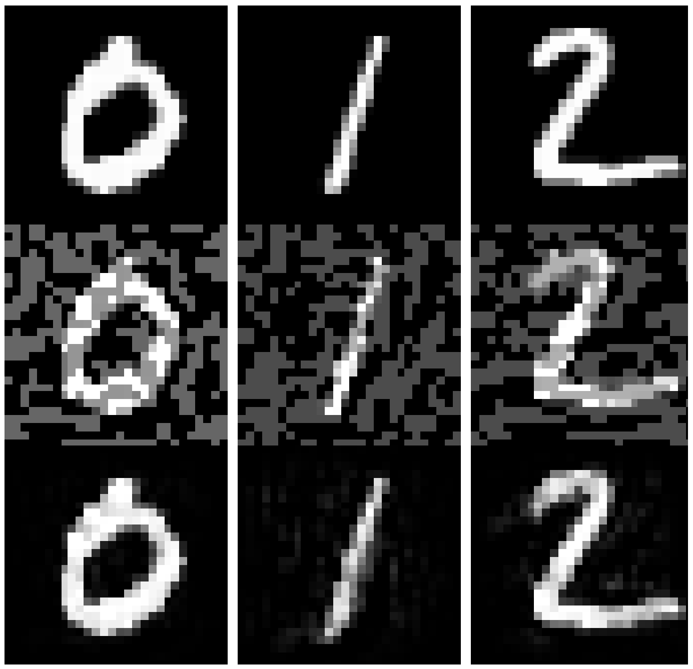


Figure 5.6: Row 1: Original images, Row 2: Perturbed images generated by FGSM [8], Row 3: Perturbed images generated by Targeted DeepFool.

Attack	Ground-truth	Target	Predicted	L_2-norm
FGSM [8]	0	–	5	23510.21
	1	–	7	15058.96
	2	–	6	19755.16
Targeted DeepFool	0	5	5	17699.60
	1	7	7	9738.00
	2	6	6	16982.90

Table 5.5: L_2-norm between the original image and the adversarial sample generated by FGSM [8] and Targeted DeepFool.

CHAPTER 6
Conclusion

6.1 Conclusion and Summary

Targeted adversarial attacks show remarkable utility for testing applications such as person recognition. In this work, we proposed a targeted adversarial attack called Targeted DeepFool with a motive to compute minimal amount of perturbation to fool a well trained deep neural network. In process of devising the algorithm, we first designed the basic algorithm for Targeted DeepFool. The basic algorithm eventually faced an issue of input getting stuck on intermediate hyperplanes. This issue is resolved by our recursive version of the algorithm. The performance of recursive Targeted DeepFool is validated by the experiments performed on grayscale as well as RGB channel data. The proposed algorithm is found to be effective in terms of adversarial success rate and average distortion, thereby adding to the properties of untargeted DeepFool algorithm.

6.2 Future Work

As a part of future work,

- this research can be extended for black box attack by making an ensemble of multiple DNNs.
- the concept of transferability can be introduced after converting it to a black box attack.
- the algorithm can be improved further to validate it on large datasets and face image data.

6.3 Publication

Shivangi Gajjar, Avik Hati, Shruti Bhilare, and Srimanta Mandal, "Generating Targeted Adversarial Attacks and Assessing their Effectiveness in Fooling Deep Neural Networks", IEEE International Conference on Signal Processing and Communications (SPCOM), Bangalore, July 2022 (Accepted).

References

[1] Y. Bi, B. Fan, and F. Wu. Beyond mahalanobis metric: cayley-klein metric learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2339–2347, 2015.

[2] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017.

[3] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 15–26, 2017.

[4] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[5] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[6] Y. Dong, T. Pang, H. Su, and J. Zhu. Evading defenses to transferable adversarial examples by translation-invariant attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[7] A. Goel, A. Singh, A. Agarwal, M. Vatsa, and R. Singh. Smartbox: Benchmarking adversarial detection and mitigation algorithms for face recognition. In *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, pages 1–7. IEEE, 2018.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[9] H. Karimi and J. Tang. Decision boundary of deep neural networks: Challenges and opportunities. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 919–920, 2020.

[10] Y. Kim, J. W. Soh, G. Y. Park, and N. I. Cho. Transfer learning from synthetic to real-noise denoising with adaptive instance normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3482–3492, 2020.

[11] A. Krizhevsky, V. Nair, and G. Hinton. CIFAR-10 (Canadian Institute for Advanced Research). *URL https://www.cs.toronto.edu/ kriz/cifar.html*, 5(4):1, 2010.

[12] C. Laidlaw and S. Feizi. Playing it safe: Adversarial robustness with an abstain option, 2019.

[13] M. Li, C. Deng, T. Li, J. Yan, X. Gao, and H. Huang. Towards transferable targeted attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[14] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[15] E. Mathieu, C. Le Lan, C. J. Maddison, R. Tomioka, and Y. W. Teh. Continuous hierarchical representations with poincaré variational auto-encoders. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[16] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[17] N. Narodytska and S. P. Kasiviswanathan. Simple black-box adversarial perturbations for deep networks, 2016.

[18] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

[19] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2(11):e7, 2017.

[20] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 372–387, 2016.

[21] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

[22] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks, 2014.

[23] J. Vadillo, R. Santana, and J. A. Lozano. Exploring gaps in deepfool in search of more effective adversarial perturbations. In G. Nicosia, V. Ojha, E. La Malfa, G. Jansen, V. Sciacca, P. Pardalos, G. Giuffrida, and R. Umeton, editors, *Machine Learning, Optimization, and Data Science*, pages 215–227, Cham, 2020. Springer International Publishing.

[24] J. Wu. Generating adversarial examples in the harsh conditions, 2020.

[25] C. Xie, Z. Zhang, Y. Zhou, S. Bai, J. Wang, Z. Ren, and A. L. Yuille. Improving transferability of adversarial examples with input diversity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[26] J. Xu, Z. Cai, and W. Shen. Using FGSM targeted attack to improve the transferability of adversarial example. In *IEEE 2nd International Conference on Electronics and Communication Engineering (ICECE)*, pages 20–25, 2019.

[27] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan. Theoretically principled trade-off between robustness and accuracy. In *International conference on machine learning*, pages 7472–7482. PMLR, 2019.

[28] Y. Zhang, Y. Song, J. Liang, K. Bai, and Q. Yang. Two sides of the same coin: White-box and black-box attacks for transfer learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '20, page 2989–2997, New York, NY, USA, 2020. Association for Computing Machinery.