Article

Beetle Antennae Search: Using Biomimetic Foraging Behaviour of Beetles to Fool a Well-Trained Neuro-Intelligent System

Ameer Hamza Khan [1], Xinwei Cao [2,*], Bin Xu [3] and Shuai Li [4]

[1] Smart City Research Institute, The Hong Kong Polytechnic University, Kowloon 999077, Hong Kong; ahakhan@polyu.edu.hk
[2] School of Business, Jiangnan University, Wuxi 214122, China
[3] School of Automation, Northwestern Polytechnical University, Xi'an 710072, China; mileface.binxu@gmail.com
[4] School of Informatics, Lanzhou University, Lanzhou 730000, China; lishuai@lzu.edu.cn
* Correspondence: xcao.fudan@gmail.com

Abstract: Deep Convolutional Neural Networks (CNNs) represent the state-of-the-art artificially intelligent computing models for image classification. The advanced cognition and pattern recognition abilities possessed by humans are ascribed to the intricate and complex neurological connection in human brains. CNNs are inspired by the neurological structure of the human brain and show performance at par with humans in image recognition and classification tasks. On the lower extreme of the neurological complexity spectrum lie small organisms such as insects and worms, with simple brain structures and limited cognition abilities, pattern recognition, and intelligent decision-making abilities. However, billions of years of evolution guided by natural selection have imparted basic survival instincts, which appear as an "intelligent behavior". In this paper, we put forward the evidence that a simple algorithm inspired by the behavior of a beetle (an insect) can fool CNNs in image classification tasks by just perturbing a single pixel. The proposed algorithm accomplishes this in a computationally efficient manner as compared to the other adversarial attacking algorithms proposed in the literature. The novel feature of the proposed algorithm as compared to other metaheuristics approaches for fooling a neural network, is that it mimics the behavior of a single beetle and requires fewer search particles. On the contrary, other metaheuristic algorithms rely on the social or swarming behavior of the organisms, requiring a large population of search particles. We evaluated the performance of the proposed algorithm on LeNet-5 and ResNet architecture using the CIFAR-10 dataset. The results show a high success rate for the proposed algorithms. The proposed strategy raises a concern about the robustness and security aspects of artificially intelligent learning systems.

Keywords: fooling attacks; nature-inspired algorithm; cognitive intelligence; neuro-intelligent systems

check for updates

Citation: Khan, A.H.; Cao, X.; Xu, B.; Li, S. Beetle Antennae Search: Using Biomimetic Foraging Behaviour of Beetles to Fool a Well-Trained Neuro-Intelligent System. *Biomimetics* **2022**, *7*, 84. https://doi.org/10.3390/biomimetics7030084

Academic Editor: Stanislav N. Gorb

Received: 30 May 2022
Accepted: 16 June 2022
Published: 23 June 2022

1. Introduction

The level of intelligence, learning, and cognitive ability possessed by living beings is directly associated with the structural complexity [1,2] and the size of the brain [3–5]. Human brains demonstrate an excellent ability to discover, learn, and recognize intricate patterns because of complex interconnection between neurons [6]. The complexity of the neural structure of the human brain has inspired the development of artificially intelligent learning systems [7–11], which are rapidly coming at par with the performance of human brain itself [12–14]. As shown in Figure 1, on the other end of the biological intelligence spectrum, lies small organisms e.g., worms and insects, which demonstrate an rudimentary-intelligent behavior and limited learning ability [15] because the neurological structure is not complex enough to develop intelligent reasoning. However, in this paper, we demonstrate a counter-intuitive result that an artificial learning system, inspired by the human brain, can be fooled by following a straightforward algorithm inspired by the behavior of an insect. More specifically, the proposed algorithm is inspired by the Beetle Antennae Search (BAS) algorithm based on the food foraging behavior of beetles [16–19]. This simple algorithm can successfully fool a deliberately designed and extensively trained Convolutional Neural Networks (CNNs) to make the wrong decisions for recognizing objects in input images. We anticipate that the presented results will provide further insight into the nature of artificial learning systems. Furthermore, these results create important implications for the security and robustness of the intelligent computation models and how it will affect the future of artificial intelligence, particularly since such systems are being widely used in commercial products [20–24].

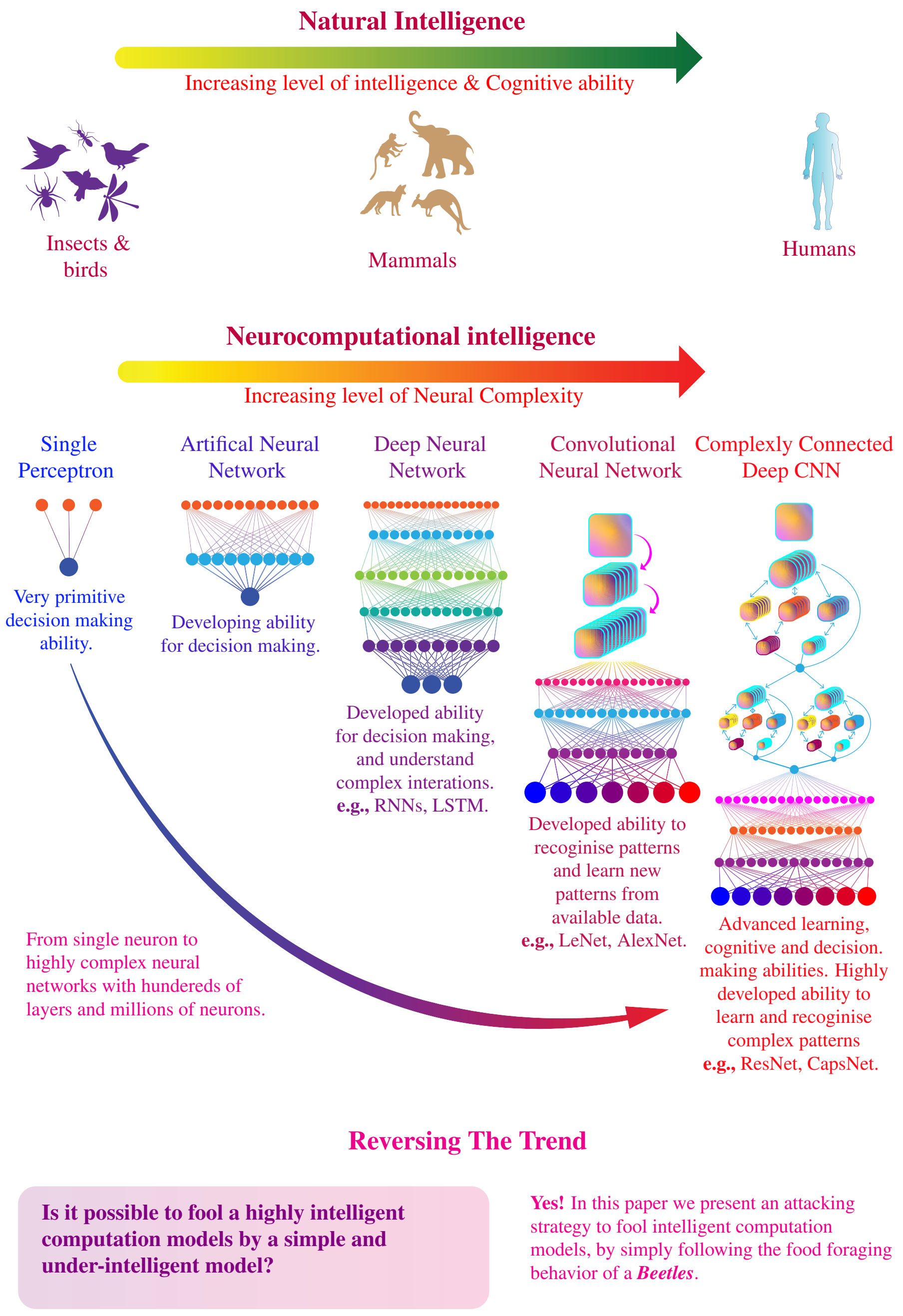

Natural Intelligence
Increasing level of intelligence & Cognitive ability
Insects & birds
Mammals
Humans
Neurocomputational intelligence
Increasing level of Neural Complexity
Single Perceptron
Artifical Neural Network
Deep Neural Network
Convolutional Neural Network
Complexly Connected Deep CNN
Very primitive decision making ability.
Developing ability for decision making.
Developed ability for decision making, and understand complex interations. e.g., RNNs, LSTM.
Developed ability to recognise patterns and learn new patterns from available data. e.g., LeNet, AlexNet.
Advanced learning, cognitive and decision. making abilities. Highly developed ability to learn and recoginise complex patterns e.g., ResNet, CapsNet.
From single neuron to highly complex neural networks with hundereds of layers and millions of neurons.
Reversing The Trend
Is it possible to fool a highly intelligent computation models by a simple and under-intelligent model?
Yes! In this paper we present an attacking strategy to fool intelligent computation models, by simply following the food foraging behavior of a Beetles.

Figure 1. Evolution of artifical and neurocomputational intelligence. The spectrum of natural intelligence ranges from simple organisms to highly intelligent human brains. The spectrum of artificial neurocomputational intelligence also follows a similar trend in term of complexity and ranges from single perceptron to multi-layered neural networks comprising of millions of neurons.

The proposed algorithm belongs to the class of nature-inspired metaheuristic algorithms. In recent years, nature-inspired optimization has been an active area of research and has given rise to several efficient optimization algorithms [11,19,25]. For example, biological evolution and natural selection have inspired the formulation of Evolutionary and Genetic Algorithms (GAs) [26]. Similarly, the flocking and swarming behavior (also referred to as swarm intelligence) of birds has inspired the creation of Particle Swarm Optimization (PSO). Similarly several other algorithms have been proposed inspired by social behaviour of other animals and insects, e.g., Ant Colony Optimization (ACO) [27], Artificial Fish Swarm Algorithm [28], Cuckoo Search [29], Invasive Weed Optimization (IWO) [30], Honey Bee Algorithm (HBA) [31], and Firefly Algorithms (FAs) [32], Grey Wolf Optimizer (GWO) [33], Ref Fox Optimizer (RFO) [34,35], and Black Widow Optimization (BWO) [36,37]. The interesting difference between BAS and other nature-inspired algorithms is that it isn't based on the swarming or social behavior of the animals. It is inspired by the food foraging behavior of a single beetle. Beetle behavior is of particular research interest because, unlike other insects, beetles usually do not work in a swarm and have the ability to search for food individually. It requires fewer search particles and objective function evaluations in each iteration of the algorithm, making it computationally efficient. Table 1 shows comparison between different metaheuristic algorithms proposed in literature for fooling neural network models.

Table 1. Comparison between metaheuristic algorithms proposed in the literature for fooling Neural Network Models.

Algorithm	Nature-Inspired	Attacked Model	Dataset Type	Number Search Particles
Grey wolf optimization [38]	Yes	AlexNet	Image sequences	Several
SIGMA [39]	No	Neural Networks	Network Intrusion detection dataset	30
PSO [40]	Yes	BiLSTM and BERT	Text (Natural Language)	8
Differential Evolution [41]	No	VGG16 and AlexNet	Images (CIFAR-10)	400
BAS (proposed)	Yes	LeNet and ResNet	Images (CIFAR-10)	2

Deep CNNs are a special type of neural network, designed specifically for learning the features from images. CNNs make use of spatial correlation of image pixels and therefore match the mechanism of how the human brain process visual signal. CNNs have been an active area of research in recent years [14,42,43] and are able to outperform the traditional processing algorithms in several tasks such as, image classification [44], segmentation [45], and restoration [46]. However, several studies also approached the CNNs from the perspective of security, robustness, and sensitivity to the noise in the input image. Szegedy et al. [47] highlight the sensitivity of the CNN to different factors of the input image, e.g., colors, light direction, the posture of objects, and several other factors. Based on this, several algorithms have been proposed [48] to fool CNN to misclassify an input image. Most of these algorithms work by adding a properly designed perturbation to the input image such that the CNN will eventually misclassify it. Goodfellow et al. [41,48–50] proposed a mechanism to calculate the required perturbation to fool a CNN by exploiting their linear behavior in high-dimensional space. Similarly, Moosavi-Dezfooli et al. [49] proposed DeepFool, an optimization-based algorithm to find the required perturbation. Su et al. [41] extended the scope of attacking strategies by fooling a CNN by perturbing just a single pixel. They used a population-based metaheuristic algorithm, called differential evolution, to find an optimal image perturbation. However, these strategies either require the estimation of the gradient of the objective function or use a population of several searching particles, which make them computationally expensive to execute.

In this paper, we propose a computationally efficient searching strategy based on the nature-inspired food foraging behavior of beetle to fool a CNN by just modifying a single pixel. Nature-inspired optimization algorithms [51–53] have been popular in the literature because of their ability to solve complex nonlinear optimization problems without relying on the gradient of the objective function [54–58]. They show efficient computational properties and fast convergence performance [59–62]. First, we mathematically formulate the CNN fooling as a constrained optimization problem. The objective function takes the pixel location, and brightness value as input, and its value is proportional to the confidence of CNN in the real class. The solution to the optimization problem will minimize the confidence of CNN in a real class, such that it will output the wrong class. Next, we propose the optimization algorithm, which uses a single search particle to search for the pixel-perturbation, to fool the CNN.

It should be noted that in contrast to the current algorithm, the proposed algorithm neither requires the gradient of the objective function nor the use of a population of several search particles. In fact, it just uses a single search particle, which makes it much more numerically efficient for a large-scale attack on a CNN. This work is motivated by the need to highlight the weakness of artificial learning systems. Our aim is to motivate the development of intelligent computation models with robustness as a primary concern in addition to accuracy [43]. The main highlights of this paper are listed as follows:

1. A pixel-level fooling attack algorithm for CNN, by just using a single search particle.
2. The algorithm is independent of the architecture of the CNN. It treats the CNN as a black-box and just relies on the output prediction of the CNN. Therefore, the algorithm is general enough to be applied to different CNN architectures.
3. The algorithm is very efficient since it relies on only a single search particle.
4. Extensive experimental results using two CNN architectures; LeNet-5 and ResNet are presented to demonstrate the efficacy of the proposed algorithm.

The rest of the paper is organized as follows. Section 2 will present the problem formulated and mathematically model the CNN fooling as a constrained optimization problem. Section 3 will formulate the algorithm to solve the formulated optimization problem. Section 4 presents the methodology to evaluate the performance of the proposed algorithm, along with the experimental results. Section 5 concludes the paper.

2. Problem Formulation

In this section, we will present the mathematical formulation of the optimization problem used to conduct a pixel-level attack on the CNN. First, we briefly describe the structure of CNNs, then discuss two types of attacks; targeted and untargeted.

2.1. Convolutional Neural Networks

CNNs belong to a special type of neural networks, designed specifically for learning the features on visual data, e.g., images. The architecture of CNNs allows taking advantage of spatial correlation between a neighboring pixel of the images, a feature lacking from a simple feedforward neural network. In CNNs, the input of the network is followed by convolutional layers and downsampling layers, before the fully connected layers, to extract useful image features efficiently. A typical CNN is shown in Figure 2. It is a simple CNN architecture with a convolutional layer, a downsampling layer, and a fully connected layer followed by a softmax layer to output probabilities of the image belonging to a particular class.

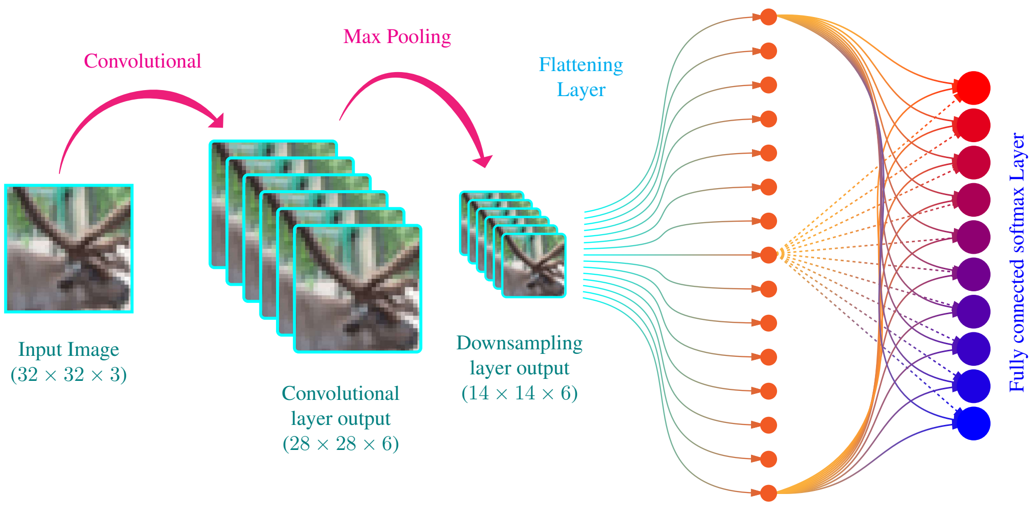

Convolutional
Max Pooling
Flattening Layer
Input Image (32 × 32 × 3)
Convolutional layer output (28 × 28 × 6)
Downsampling layer output (14 × 14 × 6)
Fully connected softmax Layer

Figure 2. A simple structure of CNN for an image classification task. The network consists of one convolutional layer, one downsampling layer, and a fully connected layer followed by the output layer. The network output the probabilities that the image contains particular objects.

The most common application of CNN is image classification. It takes an image as the input, and assign probabilities to each output class. The probabilities represent the confidence of the neural network, whether the input image belongs to a particular class. To write it mathematically, a CNN can be considered a nonlinear function f_{cnn} applied on an image input $\mathbf{X}_{img}$, which return the probability for each class

$$\mathbf{p} = f_{cnn}(\mathbf{X}_{img}), \tag{1}$$

where $\mathbf{p}$ is the vector of output probabilities, with each element corresponding to an output class. Suppose the set of all classes is represented by a vector $\mathbb{C}$,

$$\mathbb{C} = [C_1,\ C_2,\ \ldots,\ C_N]^T \in \mathbb{R}^N \tag{2}$$

where N is the total number of output classes and $C_1,\ C_2,\ \ldots,\ C_N$ are the classes labels e.g., car, aeroplane, cat, dog, etc.

Since CNN outputs a probability for each class, the class with the highest probability is called the prediction of the neural network,

$$\begin{aligned} i^* &= \underset{i}{\text{argmax}}\ \ \mathbf{p}[i], \\ C^* &= \mathbb{C}[i^*], \end{aligned} \tag{3}$$

where argmax_i returns the index of the element of $\mathbf{p}$ with maximum value, [.] represent indexing operator into a vector and C^* is the label of the predicted class.

Based on (1) and (3), it can be infered that the predicted class $\mathbb{C}^*$ is actually a function of input image $\mathbf{X}_{img}$, therefore we can write

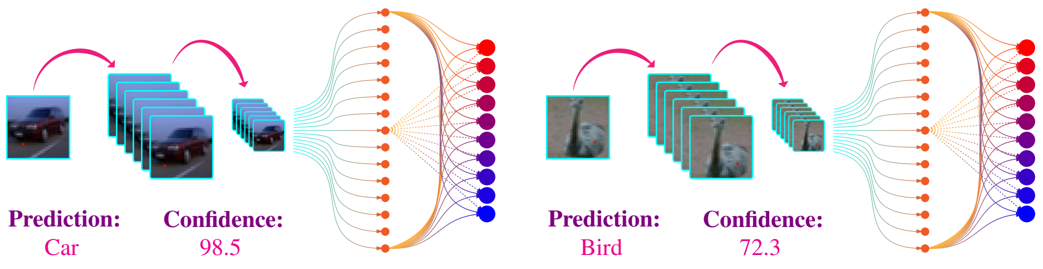
$$\mathbb{C}^* = \mathbb{F}(\mathbf{X}_{img}). \tag{4}$$

where $\mathbb{F}(.)$ represent the combined effect of (1) and (3). Figure 3 shows the predictions of simple CNN architecture introduced in Figure 2 on real-world images.

For simplification of mathematical notation in later sections, let us define a new function $\mathbb{P}$ as follow

$$\mathbb{P}(\mathbf{p}, C_i) = \mathbf{p}[i], \tag{5}$$

which takes probability vector $\mathbf{p}$ and class name C_i as input and outputs the probability of the corresponding class. Here $[.]$ is the index operator and gives the ith element of vector $\mathbf{p}$.



(**a**) (**b**)

Figure 3. Prediction of a trained simple CNN network on real-world images. (**a**) The network predict that the input image is a car with a confidence of 98.5%. (**b**) The network predict that the input image is a car with a bird of 98.5%.

2.2. Image Perturbation

Our CNN fooling attack works by perturbing a finite number of pixels of the input image to change the output probabilities of the neural networks. In this paper, we used a k-pixel attack strategy in which only k pixels of the input image is allowed to be changed. Higher the value of k, easier it is to fool the CNN since we can just modify a large part of the image. The real challenge lies in successfully fooling the CNN using fewer pixels e.g., single-pixel ($k = 1$) attack, where the attacking algorithm is only allowed to modify a single pixel.

To mathematically define the image perturbation function $\mathcal{P}$, consider the input image $\mathbf{X}_{img}$ has $m \times n$ rgb-pixels. The pixel intensity varies from 0 to 1 i.e., $rgb \in [0, 1]$. The perturbation function takes three metrices with k numbers of rows as input; rows $r \in [1, 2, \ldots, m]^{k\times1}$, columns $c \in [1, 2, \ldots, n]^{k\times1}$ and rgb values $rgb \in [0, 1]^{k\times3}$. The function $\mathcal{P}$ returns a perturbed image $\mathbf{X}_{per}$ such that the rows and columns provided in metrices r and c and changed with corresponding values in rgb matrix

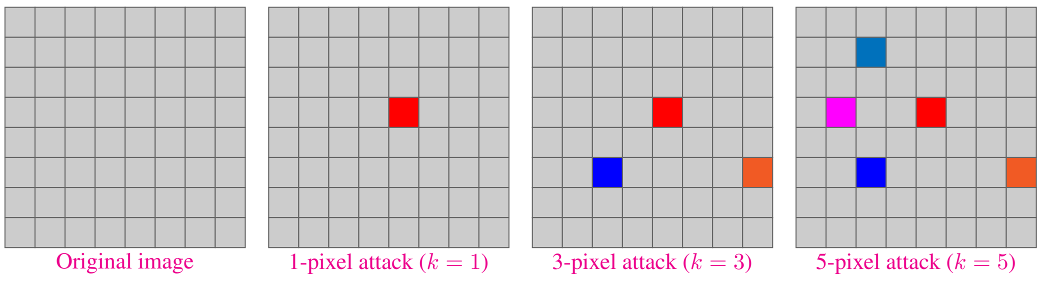
$$\mathbf{X}_{per} = \mathcal{P}(\mathbf{X}_{img},\ r,\ c,\ rgb). \tag{6}$$

The working of perturbation function $\mathcal{P}$ is defined as follow:

$$\mathcal{P} : \mathbf{X}_{img}[r[i], r[j]] := rgb[i],\ \text{where } i \in \{1, 2, \ldots, k\} \tag{7}$$

where $:=$ is the assignment operator. Figure 4 shows an illustration of k-pixel attack.

The objective of a CNN fooling attack is to find a perturbed image $\mathbf{X}_{per}$ such that the CNN outputs a wrong predicted class $C^* \neq C_{real}$. Two different types of attacks have been proposed; untargeted and targeted attacks. Now we will model each type of these attacks as an optimization problem.



(**a**) (**b**) (**c**) (**d**)

Figure 4. Illustration of k-pixel perturbation technique for attacking a CNN. It shows a image with a grid of 8×8 pixels. (**a**) Original Image, (**b**) $k = 1$, (**c**) $k = 3$, and (**d**) $k = 5$. A pixel is assigned a random RGB value for attack.

2.3. Untargeted Attack

In an untargeted attack, the objective is as follows; to change the input image $\mathbf{X}_{img}$ to a perturbed image $\mathbf{X}_{per}$ such that the predicted class C^* is not equal to the real class C_{real}. In an untargeted attack, we are not concerned about the value of the new predicted class. In other words, we just want to minimize the confidence of the CNN in the real class C_{real}. By using the definition from (5) and (1) we can write the following minimization problem

$$\mathbf{X}^*_{per} = \underset{\mathbf{X}_{per}}{\text{argmin}}\ \mathbb{P}(f_{cnn}(\mathbf{X}_{per}), C_{real}),$$

where $\mathbf{X}^*$ is the modified image which minimize the confidence of the CNN in the real class C_{real}. For k-pixel attack, (6) can be used to rewrite the above objective function as

$$\begin{aligned} r^*, c^*, rgb^* = \underset{r,c,rgb}{\text{argmin}}\ & \mathbb{P}(f_{cnn}(\mathcal{P}(\mathbf{X}_{img}, r, c, rgb)),\ C_{real}) \\ \text{Subject to:} & \\ & 0 \le r[i,1] \le m,\ \text{ where }\ i \in \{1,2,\dots,k\} \\ & 0 \le c[i,1] \le n,\ \text{ where }\ i \in \{1,2,\dots,k\} \\ & 0 \le rgb[i,j] \le n,\ \text{ where }\ i \in \{1,2,\dots,k\},\ j \in \{1,2,3\} \end{aligned} \tag{8}$$

where notation $[i,j]$ represents indexing into a matrix and returns element at ith row and kth column. r^*, c^* and rgb^* are the required rows, columns and rgb values.

2.4. Targeted Attack

In a targeted attack, the objective of the attack is not just to change the output of the network from real class C_{real}, but also fool the network to output a target class $C_{target} \neq C_{real}$. The targeted attack adds a secondary requirement on the success criteria. To model it as an optimization problem, we need to use the fact that our objective is to increase the confidence of the CNN in the target class C_{target}. We can model it as following maximization problem.

$$\mathbf{X}^*_{per} = \underset{\mathbf{X}_{per}}{\text{argmax}}\ \mathbf{P}(f_{cnn}(\mathbf{X}_{per}), C_{target}),$$

which is equivalent to the following problem in term of k-pixel attack,

$$\begin{aligned} r^*, c^*, rgb^* = \underset{r,c,rgb}{\text{argmax}}\ & \mathbf{P}(f_{cnn}(\mathcal{P}(\mathbf{X}_{img}, r, c, rgb)),\ C_{target}) \\ \text{Subject to:} & \\ & 0 \le r[i,1] \le m,\ \text{ where }\ i \in \{1,2,\dots,k\} \\ & 0 \le c[i,1] \le n,\ \text{ where }\ i \in \{1,2,\dots,k\} \\ & 0 \le rgb[i,j] \le 1,\ \text{ where }\ i \in \{1,2,\dots,k\},\ j \in \{1,2,3\} \end{aligned} \tag{9}$$

To use the same optimization framework for targeted and untargeted attacks, we can convert the above maximization problem to the following equivalent optimization problem,

$$r^*, c^*, rgb^* = \underset{r,c,rgb}{\text{argmin}}\ 1 - \mathbf{P}(f_{cnn}(\mathcal{P}(\mathbf{X}_{img}, r, c, rgb)),\ C_{target}), \tag{10}$$

where r^*, c^* and rgb^* are the rows, columns and rgb to change the output of the CNN to the targeted class.

3. Algorithm

In this section, we will present the algorithm to efficiently solve the optimization problem formulated in Section 2. First, we characterize the food foraging behavior of beetles and then mathematically formulate the algorithm.

3.1. Mathematically Modelling Behavior of Beetle

To formulate a computationally efficient optimization algorithm, requiring just a single search particle, we propose a nature-inspired optimization algorithm based on the food foraging behavior of beetles. It is in contrast to the traditional CNN fooling algorithms that require several search particles to fool the CNN. The Beetles have antennae-like structure attached to their heads. The antennae help them probe for the smell of food in the environment. This ability of using antennae to search for food is particularly interesting because it can efficiently explore an unknown environment and find the goal, i.e., food source, without any advanced sensory capabilities.

The problem mentioned above of searching for an food source is essentially an optimization problem. The distance from the food source is equivalent to the objective function. The goal of a beetle is to minimize the distance between itself and the food source. A beetle starts from a random location in the environment. At each step during the search, it uses antennae to probe its surrounding locations, i.e., calculate the value of the objective function at the location of antennae. The beetle estimate this value using the intensity of smell. Based on the probed values, it determines an incrementally favorable search direction toward the food source. It keeps on taking steps using the same strategy until finally reaching the optimal point of the objective function, i.e., food source. Figure 5 illustrates the concept.

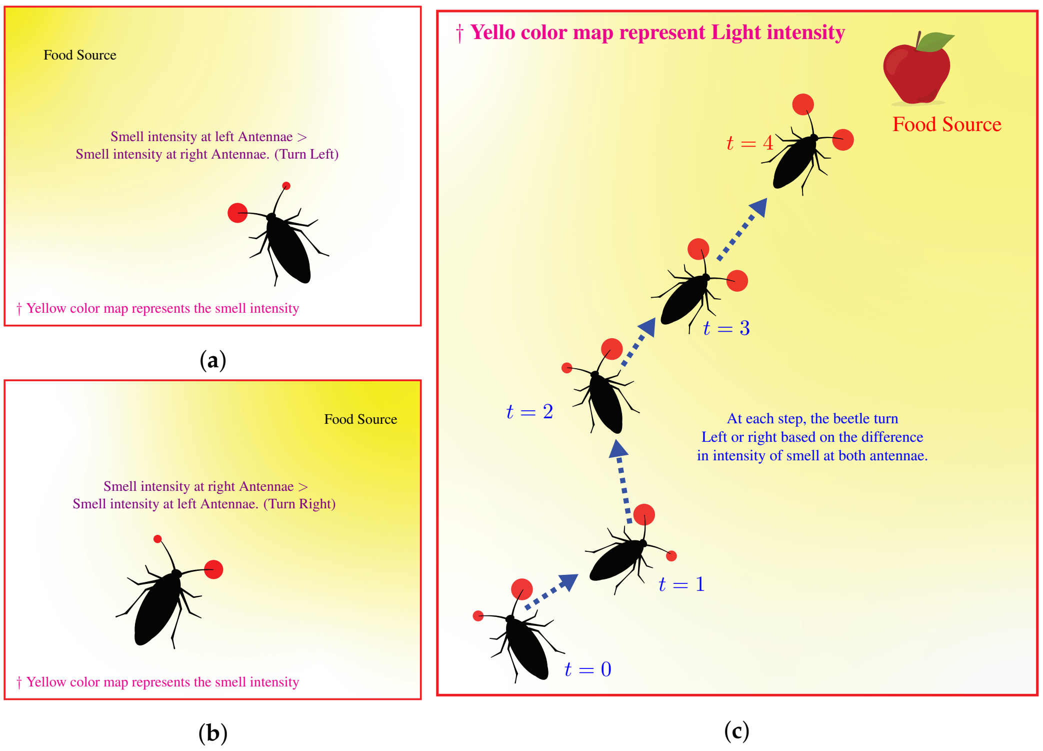

Food Source
Smell intensity at left Antennae >
Smell intensity at right Antennae. (Turn Left)
† Yellow color map represents the smell intensity
(a)
Food Source
Smell intensity at right Antennae >
Smell intensity at left Antennae. (Turn Right)
† Yellow color map represents the smell intensity
(b)
† Yello color map represent Light intensity
Food Source
$t = 4$
$t = 3$
$t = 2$
$t = 1$
$t = 0$
At each step, the beetle turn
Left or right based on the difference
in intensity of smell at both antennae.
(c)

Figure 5. Illustration of Beetle's strategy during searching for food source, i.e., maximum intensity of food smell. (**a**,**b**) shows different configuration of food source and the beetle. (**c**) Illustration of the locomotion of the microorganism.

3.2. Optimization Algorithm

To mathematically model the behavior of a beetle, let us consider the searching for the food source (i.e., place with maximum intensity of food smell) as an optimization problem. The map of smell intensity in the environment corresponds to the value of objective functions. The goal for beetle is to find the maxima of the smell intensity i.e., the food source. Let $g(\mathbf{x})$ is the function representing the smell intensity at point $\mathbf{x}$. The searching for maximum smell intensity is equivalent to the solution of following optimization problem with linear inequality constraints

$$\begin{aligned} &\max_{\mathbf{x}}\ g(\mathbf{x}) \\ &\text{Subject to: } \mathbf{x}_{\min} \le \mathbf{x} \le \mathbf{x}_{\max}. \end{aligned} \tag{11}$$

Suppose at time instant t, the beetle finds itself at position $\mathbf{x}_t$. The intensity of smell is given by $g(\mathbf{x}_t)$, at the current location. In order to take the next step, the beetle measure intensity of smell in each direction using each of its antennae. Suppose the antennare is located in the radially opposite direction, and randomly generated $\vec{b}$ represents the direction vector of left antennae relative to the current position of the beetle $\mathbf{x}_t$. The following equations give the position of antennae endpoints

$$\begin{aligned} \mathbf{x}_l &= \mathbf{x}_t + \lambda\vec{b}, \\ \mathbf{x}_r &= \mathbf{x}_t - \lambda\vec{b}, \end{aligned} \tag{12}$$

where λ is the length of an antennae, $\mathbf{x}^l$, and $\mathbf{x}^r$ are the position vectors of left and right antennae respectively. However, these vectors may violate the constraint of (11) because of randomly generated vector $\vec{b}$. Therefore, we define a constrained set Ψ as follow

$$\Psi = \{\mathbf{x} | \mathbf{x}_{\min} < \mathbf{x} < \mathbf{x}_{\max}\}.$$

and project the vectors $\mathbf{x}_l$ and $\mathbf{x}_r$ on the constrained set Ψ

$${}^{\Psi}\mathbf{x}_l = P_{\Psi}(\mathbf{x}_l), \quad {}^{\Psi}\mathbf{x}_r = P_{\Psi}(\mathbf{x}_r), \tag{13}$$

Ψ is written in superscript to denote that the vectors are projected on set Ψ. The function $P_{\Psi}(.)$ is called a projection function. We defined it as follow,

$$P_{\Psi}(\mathbf{x}) = \max\{\mathbf{x}_{\min}, \min\{\mathbf{x}, \mathbf{x}_{\max}\}\}. \tag{14}$$

Such a definition of projection function is simple and and computational efficient.

The smell intensities at projected antennae location is given by $g({}^{\Psi}\mathbf{x}_l)$ and $g({}^{\Psi}\mathbf{x}_r)$. By comparing these values, the beetle take next step according to the following rule,

$$\mathbf{x}_{new} = \mathbf{x}_t + \delta\ \text{sign}(g(\mathbf{x}_l) - g(\mathbf{x}_r))\vec{b}, \tag{15}$$

where the signum function sign(.) ensures that the next step is taken toward the direction of higher smell intensity. δ is the actual step-size taken by the beetle and proportional to Euclidean distance between x_{new} and x_t. After reaching the new location the beetle will re-measure the intensity of smell; if there is an improvement it will remain at the new location; otherwise, it will return to the previous location, i.e.,

$$\mathbf{x}_{t+1} = \begin{cases} \mathbf{x}_{new}, & \text{if, } g(\mathbf{x}_{new}) \ge g(\mathbf{x}_t) \\ \mathbf{x}_k, & \text{if, } g(\mathbf{x}_{new}) < g(\mathbf{x}_t) \end{cases}. \tag{16}$$

After reaching $\mathbf{x}_{t+1}$, we again generate a random direction vector $\vec{b}$ and repeat the same process until reaching the goal. Although the above algorithm is formulated for the maximization problem (11), it can be converted used for the minimization problem by modifying the update rule in (15) as

$$\mathbf{x}_{new} = \mathbf{x}_t - \delta\ \text{sign}(g(\mathbf{x}_l) - g(\mathbf{x}_r))\vec{b}. \tag{17}$$

The algorithm can be summarised as following:

1. Start from random location $\mathbf{x}_0$.
2. Generate a random direction vector $\vec{b}$ for left antennae relative to current position $\mathbf{x}_0$ of the beetle.
3. Calculate the position of left and right antennae ($\mathbf{x}^l$ and $\mathbf{x}^r$) using (12).
4. Calculate new position $\mathbf{x}_{t+1}$ using (15) and (16).
5. If reached goal position $\mathbf{x}_G$, stop. Otherwise, return to step 2.

To use this algorithm for solving the optimization problem of untargeted and targeted attack defined in (8) and (10) respectively, we define a matrix $\overline{\mathbf{X}}$

$$\overline{\mathbf{X}} = \begin{bmatrix} r & c & rgb \end{bmatrix}. \tag{18}$$

By considering the notation defined in Section 2.2, the dimension of $\overline{\mathbf{X}}$ becomes $k \times 5$. This definition of a new matrix $\overline{\mathbf{X}}$ allow us to define the objective function using a single variable. For an untargeted attack, the objective function becomes,

$$g(\overline{\mathbf{X}}) = \mathbb{P}(f_{cnn}(\mathcal{P}(\mathbf{X}_{img}, \overline{\mathbf{X}}[:,1], \overline{\mathbf{X}}[:,2], \overline{\mathbf{X}}[:,\{3,4,5\}])),\ C_{real}), \tag{19}$$

where the semicolon symbol (:) in the matrix indexing is used to denote the entire column of the matrix. Similarly, for the targeted attack, the objective function becomes,

$$g(\overline{\mathbf{X}}) = 1 - \mathbf{P}(f_{cnn}(\mathcal{P}(\mathbf{X}_{img}, \overline{\mathbf{X}}[:,1], \overline{\mathbf{X}}[:,2], \overline{\mathbf{X}}[:,\{3,4,5\}])),\ C_{target}). \tag{20}$$

Since r and c in (18) can only take integer values, we use the round function to convert floating-point values to integers. Based on these mathematical relations for the objective function of untargeted and targeted attacks, we formally present the steps of optimization algorithm in Algorithm 1.

3.3. Illustration of Attacking Algorithm

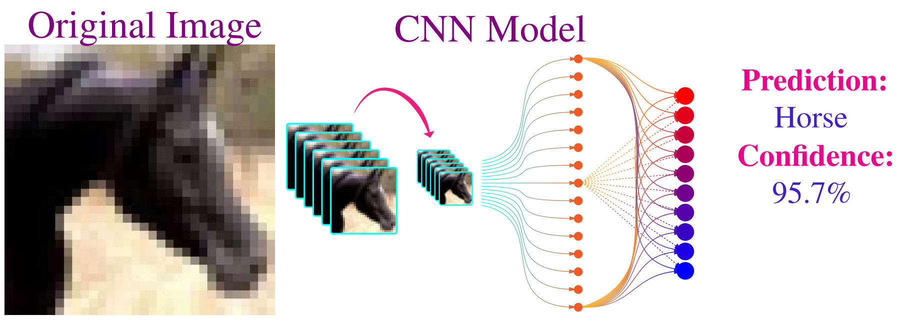
Figure 6 illustrates the proposed attacking strategy. The CNN model takes a $32 \times 32 \times 3$ RGB image as input and outputs the class of objects present in the image, i.e., horse, dog, car, etc. For the demonstration of the fooling algorithm, we used bettle with just two antennae. The original input image contains a horse. Figure 6a shows the CNN correctly predicts the class to be a horse for the original with the confidence of 95.7%. Then we start the iterations of our fooling algorithm by initializing the algorithm at a random pixel location. Figure 6b shows the modified image after the first iteration; the modified pixel is highlighted in red. At this iteration, CNN still predicts the correct class, but the confidence of reduced to 93.67%. Figure 6c shows further decrease in confidence after a mere 32 iterations. Finally, the beetle is able to find a pixel to fool the CNN after 200 iterations, as shown in Figure 6d. The network is fooled into predicting that the input image contains a cat, whereas, in reality, only one pixel of the input image is modified.

3.4. Computational Complexity

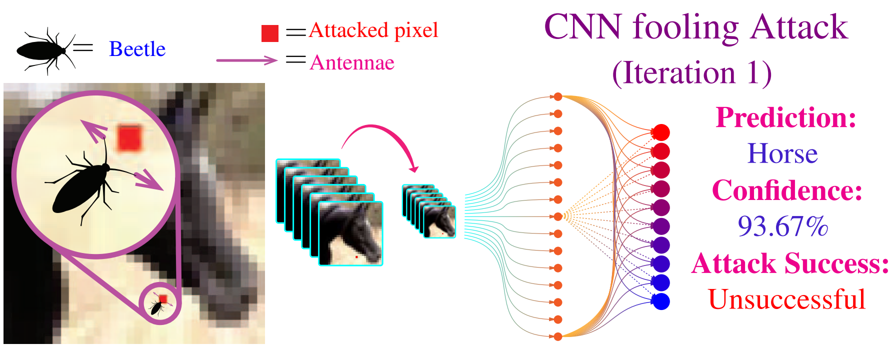
The computational complexity of the proposed algorithm can be computed by analyzing the steps listed inside while loop of Algorithm 1. The first step of the algorithm requires the generation of $(3k + k + k =)5k$ uniformly distributed random variables, where k denotes the number of attacked pixels. On modern processors with the native ability to generate random numbers, it will require $5k$ operations in the worst case (support for vectorized operation will considerably reduce this number). The second step, i.e., calculation of antennae's end-point locations, requires $(2 \times 5k =)10k$ multiplication and a similar number of additions, making up a total of $20k$ floating-point operations in this step. The next step requires the evaluation of the objective function twice. A careful analysis of the objective function tells that it basically consists of two primary steps: (1) perturbing the input image and (2) forward passing the perturbed image through the CNN. Since the neural networks are very computationally expensive systems and even a single forward pass can require a large number of floating-point operations. Although the exact numbers of floating-point operations depend on the design of CNN architecture, even small networks, such as LeNet, can require millions of computations. Let's suppose the number of floating-point operations required by the CNN model are N. Since we need to forward pass through the CNN model twice, which brings the number of floating-point operations $2N$. Adding all computations for all these steps, we reach a final value of $5k + 20k + 2N \approx 2N$. The last approximation is based on the fact that $N >> k$, i.e., the number of computations required by CNN, is much higher than the number of attacked pixels. From this analysis, it can be seen that the amount of computation per iteration is primarily dominated by the CNN model and the computation requirement of BAS are negligible in comparison.

Algorithm 1: Attacking Algorithm.

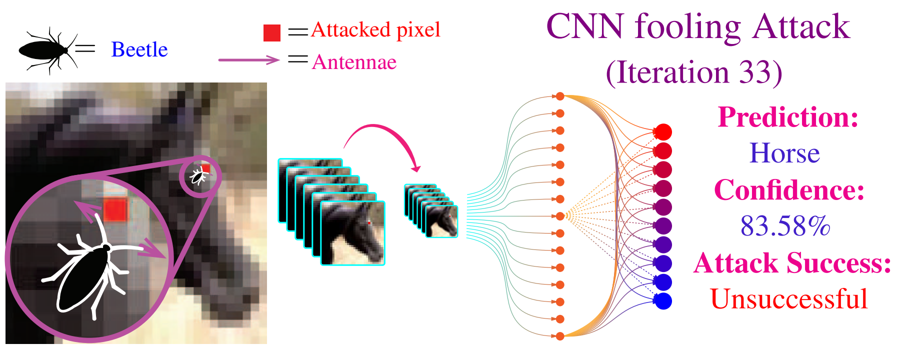
```
Input: $\mathbf{X}_{img}$, input image of dimension $m \times n$
       $\mathbb{C}$, image classes
       $\overline{\mathbf{X}}_0$, initial starting point
       $is\_targeted$, attack is targeted or untargeted
       $k$, number of attacked pixels
       $\lambda$, antennae length
       $\delta$, step-length
       $t_{stop}$, maximum number of iterations
Output: $r^*$, $c^*$ and $rgb^*$.
if $is\_targeted == True$ then
    Construct objective function $g(.)$ using (20).
else
    Construct objective function $g(.)$ using (19).
end
Define a function $rand(dim, low, upper, type)$ which generate a random matrix of
  dimension $dim$, with elements in range $[low, upper]$. The $type$ can be 'integer' or
  'float'.
$t \leftarrow 0$
$g_0 \leftarrow g(\overline{\mathbf{X}}_0)$
while $t < t_{stop}$ do
    Generate random matrices as follow:
      $\vec{b}_r \leftarrow rand((k,1), 1, m, \text{'integer'})$
      $\vec{b}_c \leftarrow rand((k,1), 1, n, \text{'integer'})$
      $\vec{b}_{rgb} \leftarrow rand((k,3), 0, 1, \text{'float'})$
    Combine the above to form the following matrix:
      $\vec{b} = [\vec{b}_r \; \vec{b}_c \; \vec{b}_{rgb}]$
    Calculate antennae locations $\overline{\mathbf{X}}_l$ and $\overline{\mathbf{X}}_r$ using (12).
      $\overline{\mathbf{X}}_l = \overline{\mathbf{X}}_t + \lambda\vec{b}, \quad \overline{\mathbf{X}}_r = \overline{\mathbf{X}}_t - \lambda\vec{b}.$
    Project the locations in the constrained set $\Psi$ using the projection function
      defined in (14).   ${}^{\Psi}\overline{\mathbf{X}}_l = P_{\Psi}(\overline{\mathbf{X}}_l), \quad {}^{\Psi}\overline{\mathbf{X}}_r = P_{\Psi}(\overline{\mathbf{X}}_r).$
    Calculate the new position for the beetle using (15).
      $\overline{\mathbf{X}}_{new} = \overline{\mathbf{X}}_t - \delta\,\text{sign}(g(\overline{\mathbf{X}}_l) - g(\overline{\mathbf{X}}_r))\vec{b}.$
    Evaluate objective function at new location:
      $g_{new} \leftarrow g(\overline{\mathbf{X}}_{new})$
    if $g_{new} < g_t$ then
        $\overline{\mathbf{X}}_{t+1} \leftarrow \overline{\mathbf{X}}_{new}, \qquad g_{t+1} \leftarrow g_{new}$
    else
        $\overline{\mathbf{X}}_{t+1} \leftarrow \overline{\mathbf{X}}_t, \qquad g_{t+1} \leftarrow g_t$
    end
    $t \leftarrow t + 1$
end
```



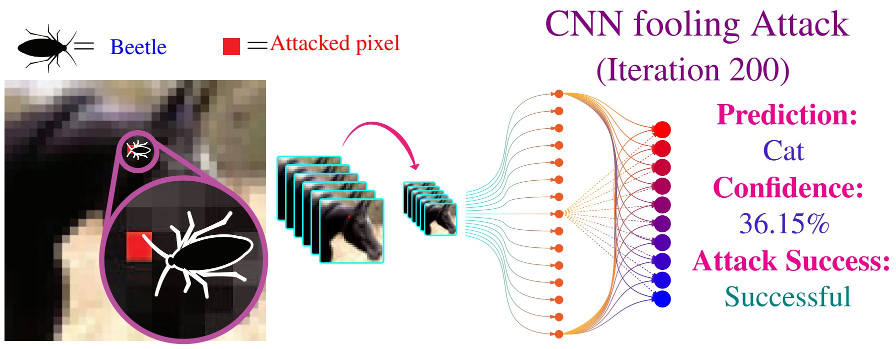
(**a**)



(**b**)



(**c**)



(**d**)

Figure 6. Illustration of the CNN attacking algorithm. (**a**), The original input image to the CNN model: the image contains a horse, and the CNN's prediction is also a horse with the confidence of 95.7%. (**b**), First iteration of the fooling attack: the modified image reduced the confidence to 93.67% with just modification of single pixel. (**c**), 33rd iteration of the fooling attack: after searching for a while, the algorithm found a pixel which reduced confidence to 83.58%. (**d**), 200th iteration of the fooling attack: the algorithm finally found a pixel which fools the CNN to classify the image as a cat.

4. Experiment Methodology and Results

In this section, we will present the experimental methodology to evaluate the success rate of the proposed attacking algorithm. Then we will discuss the results.

4.1. Evaluation Methodology

To statistically evaluate the effectiveness of the proposed algorithm we choose two CNN architectures; LeNet-5 [63] and ResNet [64], trained on CIFAR-10 dataset [65].

4.1.1. Image Dataset

The CIFAR-10 dataset consists of a total of 60,000 RGB images. Each image have a dimension of $32 \times 32 \times 3$. The images belong to 10 classes, namely airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. The dataset is evenly distributed among ten classes i.e., 6000 images belong to each class. The dataset is divided into two portions; the first portion is a training dataset, contains 50,000 images, while the second portion contains 10,000 images reserved for testing. The dataset is widely used in the training and testing of computer vision and machine learning models. The dataset is split into 50,000 training and 10,000 test images. Figure 7 shows one sample image from each of the ten classes of the dataset.

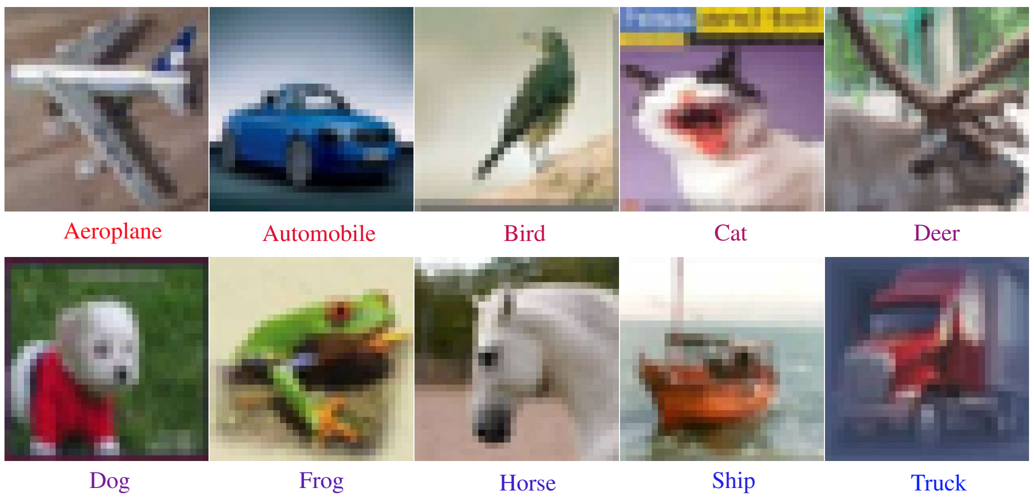


Figure 7. Sample images from the CIFAR-10 dataset. The dataset contains a total of 10 classes, including animals and objects. One image from each class is shown here.

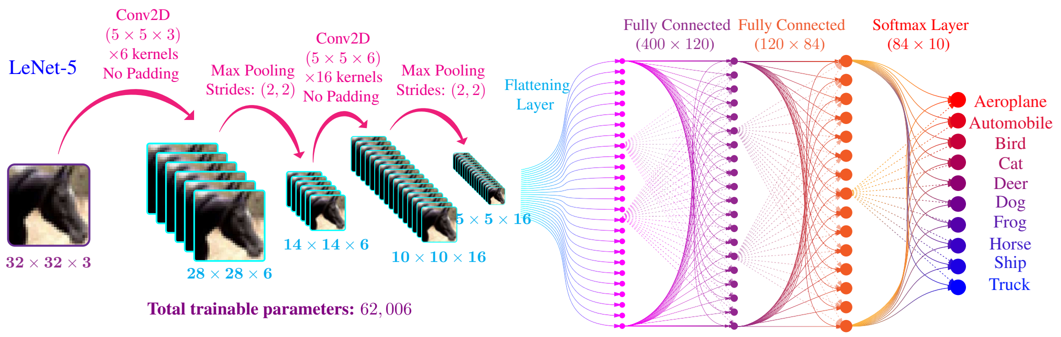
4.1.2. LeNet-5 Architecture

LeNet is a five-layer CNN architecture. It consists of two 2D-convolutional with downsampling layers, followed by two fully connected layers and the softmax output layer. The architecture of LeNet is shown in Figure 8a. The input to the LeNet is a $32 \times 32 \times 3$ RGB image from CIFAR-10 dataset and output is a 10×1 probability vector. The hidden layers of LeNet-5 are in the following order.

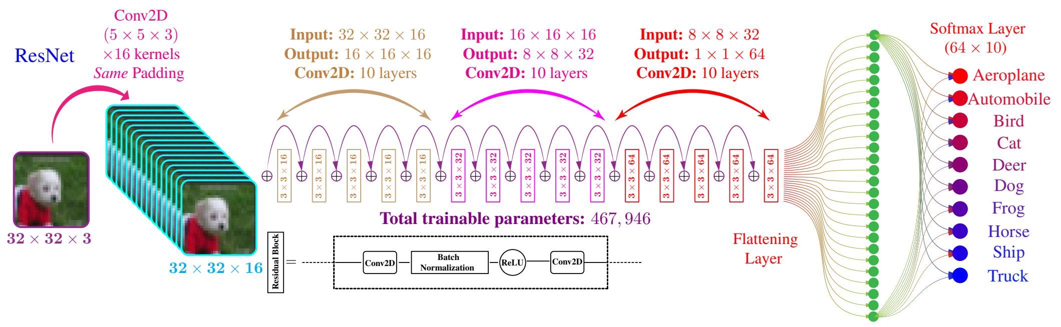
1. The first hidden layer of LeNet is a 2D convolutional layer with six kernels, each of dimension $5 \times five \times 3$. Each kernel uses a rectified linear unit (ReLU) as an activation function. The total tunable parameter in this layer, including the bias parameters, are $5 \times 5 \times 3 \times 6 + 6 = 456$. A max-pooling layer follows the convolutional layer with a stride of $(2, 2)$.
2. The second hidden layer is similar 2D convolutional layer with 16 kernels, each of dimension $5 \times 5 \times 6$. Total number of tunable parameters in this layer are $5 \times 5 \times 6 \times 16 + 16 = 2416$.
3. The output of the second convolutional layer is flattened from a $5 \times 5 \times 16$ to a 400×1 vector. The flattened layer is connected to a fully connected later with 120 neurons with ReLU activation. The total trainable parameters in this layer are $400 \times 120 + 120 = 48{,}120$.

4. The fourth layer is also a fully connected layer with a total of 84 neurons with ReLU activation. Connection with layer three makes the total trainable parameters in this layer to be $120 \times 84 + 84 = 10{,}164$.
5. The last layer is a fully-connected layer with ten neurons using softmax activation. The connection with fourth layer makes a total of $84 \times 10 + 10 = 850$ trainable parameters. The output is 10×1 vector.

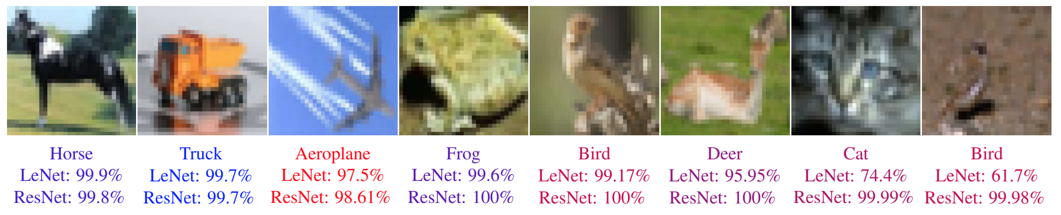
The above mentioned 5 layers make the total trainable parameter count for LeNet-5 architecture to be 62,006.


LeNet-5
Conv2D (5 × 5 × 3) ×6 kernels No Padding
Max Pooling Strides: (2, 2)
Conv2D (5 × 5 × 6) ×16 kernels No Padding
Max Pooling Strides: (2, 2)
Flattening Layer
Fully Connected (400 × 120)
Fully Connected (120 × 84)
Softmax Layer (84 × 10)
32 × 32 × 3
28 × 28 × 6
14 × 14 × 6
10 × 10 × 16
5 × 5 × 16
Total trainable parameters: 62, 006
Aeroplane
Automobile
Bird
Cat
Deer
Dog
Frog
Horse
Ship
Truck

(**a**)


ResNet
Conv2D (5 × 5 × 3) ×16 kernels Same Padding
32 × 32 × 3
32 × 32 × 16
Input: 32 × 32 × 16 Output: 16 × 16 × 16 Conv2D: 10 layers
Input: 16 × 16 × 16 Output: 8 × 8 × 32 Conv2D: 10 layers
Input: 8 × 8 × 32 Output: 1 × 1 × 64 Conv2D: 10 layers
3 × 3 × 16
3 × 3 × 32
3 × 3 × 64
Total trainable parameters: 467, 946
Residual Block
Conv2D
Batch Normalization
ReLU
Conv2D
Flattening Layer
Softmax Layer (64 × 10)
Aeroplane
Automobile
Bird
Cat
Deer
Dog
Frog
Horse
Ship
Truck

(**b**)


Horse LeNet: 99.9% ResNet: 99.8%
Truck LeNet: 99.7% ResNet: 99.7%
Aeroplane LeNet: 97.5% ResNet: 98.61%
Frog LeNet: 99.6% ResNet: 100%
Bird LeNet: 99.17% ResNet: 100%
Deer LeNet: 95.95% ResNet: 100%
Cat LeNet: 74.4% ResNet: 99.99%
Bird LeNet: 61.7% ResNet: 99.98%

(**c**)

Figure 8. CNN architecture used for experiments in this paper. (**a**) Topology of LeNet-5, it contains a total of 5 trainable layers. (**b**) Topology of ResNet, it contains a total of 32 trainable layers. (**c**) Prediction of trained LeNet-5 and ResNet models for sample images of CIFAR-10 dataset. ResNet usually show high confidence as compared to LeNet-5 because of higher number of layers.

4.1.3. ResNet Architecture

Residual Network (ResNet) architecture is another popular CNN architecture that employs the concept of residual learning to train the neural networks efficiently. The used ResNet architecture is shown in Figure 8b. It can be seen that the architecture contains a special type of block called residual blocks. Each residual block contains two 2D-convolutional layers. Apart from the usual forward path through convolutional layers, the residual blocks also have an alternate shortcut forward path. This shortcut path allows the convolutional layers only to learn residual mapping instead of actual mapping. There are a total of 15 residual blocks and $15 \times 2 + 1 = 31$ convolutional layers. Similar to LeNet-5 architecture, the ResNet takes a $32 \times 32 \times 3$ RGB image as input and output a probability vector. Some features of ResNet architecture are briefly given below.

1. The first hidden layer is a 2D convolutional layer with 16 kernels, each of dimension $3 \times 3 \times 3$. This convolutional layer uses zero paddings; therefore, the output of this layer has a dimension of $32 \times 32 \times 16$.
2. The first convolutional layer is followed by a set of 5 similar residual blocks. Each residual block contains two convolutional layers. Each of the convolutional layers contains 16 kernels of dimension 3×3 and uses zero paddings to maintain the dimension between its input and output. Each convolutional layers outputs a 3D-matrix of dimension $32 \times 32 \times 16$, except the output of fifth residual block which apply max polling with stride $(2, 2)$, making output dimension $16 \times 16 \times 16$.
3. After the fifth residual block, we have another set of 5 residual blocks. For this set of the block, each convolutional layer has a total of 32 kernels of dimension (3×3). Each convolutional layer outputs a 3D-matrix of dimension $16 \times 16 \times 32$, except the output of the tenth block, which employs a max-pooling layer and outputs a 3D-matrix of dimension $8 \times 8 \times 32$.
4. The residual blocks from eleventh to fifteenth are also similar to each other. The convolutional layers in these blocks have a total of 64 kernels of dimension $(3, 3)$. The output of each convolutional layers is a 3D-matrix of dimension $8 \times 8 \times 64$.
5. The output of the fifteenth residual block is passed through a global average pooling layer and outputs a 3D-matrix of dimension $8 \times 8 \times 1$.
6. The $8 \times 8 \times 1$ tensor is flattened into 64×1 vector and connected with a fully connected layer of 10 neurons with softmax activation.

Apart from the output layer, all the remaining layers use ReLU activation. The total number of trainable parameters in this ResNet architecture are 467,946. This number is almost 7.5 times larger then LeNet-5. Due to the higher number of parameters and more complexity, the ResNet architecture shows much better performance as compared to LeNet-5.

4.1.4. Training of CNNs

We implemented both CNN architectures in TensorFlow [66] and trained them using the 50,000 training images from the CIFAR-10 dataset using categorical cross-entropy as the loss function. Figure 8c shows the prediction of the CNN for some sample images along with the corresponding confidence values for the trained LeNet-5 and ResNet models. It can be seen that the confidence value of the ResNet model is comparatively higher than the LeNet-5 model for most of the image. It can be attributed to the larger size and complex structure of ResNet. The complex structure allows the ResNet to learn more image features as compared to LeNet-5. The prediction accuracy of the LeNet-5 and ResNet model on the training and testing models are summarized in Table 2.

Table 2. Summary of trained CNN architectures.

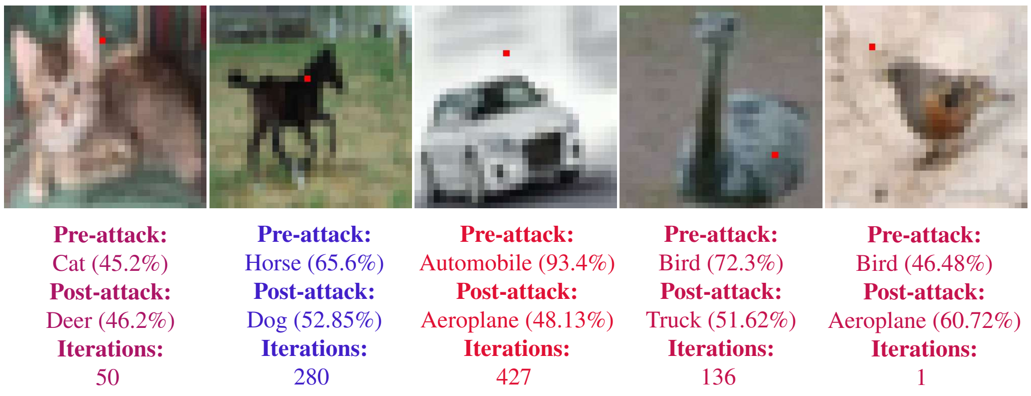
×	**No. of Parameters**	**Training Samples**	**Training Accuracy**	**Testing Samples**	**Testing Accuracy**
LeNet-5	50,000	50,000	78.47%	10,000	74.88%
ResNet	50,000	50,000	99.83%	10,000	92.31%

4.2. Results and Discussion

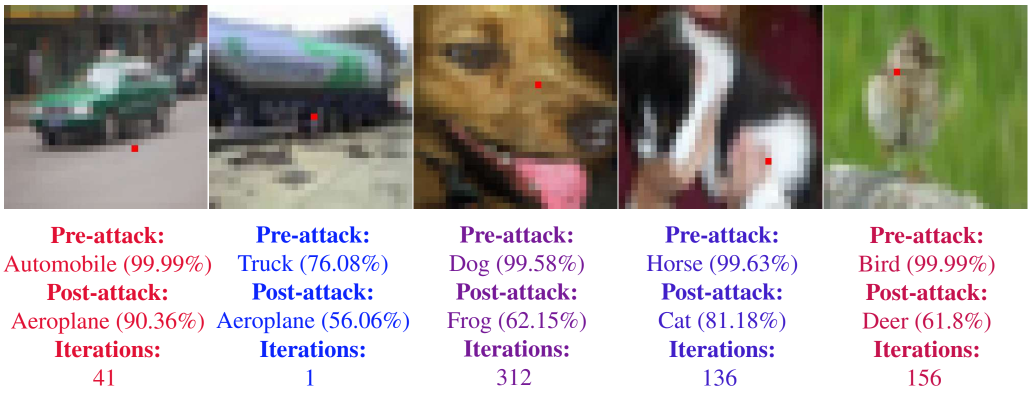
To conduct a fair evaluation of our proposed algorithm, we only attacked images from the test dataset, which are correctly classified by the corresponding trained models. According to Table 2, only 7488 out of 10,000 test images were used in fooling attack on LeNet-5. Similarly, for ResNet, 9231 out of 10,000 test images were considered. Now we will present the results for untargeted and targeted attacks on both of these networks separately.

Figure 9 shows some of the sample images for untargeted fooling attacks. Figure 9a corresponds to the images for the LeNet-5 and Figure 9b present the images for ResNet. Each image also mentions the pre-attack prediction of the corresponding CNN along with the confidence. The post-attack prediction of the wrong class, along with the new confidence value, is also shown. The modified pixel is shown as red. The number of iterations required to find the pixel to fool the CNN is also shown. An interesting observation is to note that the CNN can be fooled even when the perturbed pixel does not directly lie on the actual object.

Similarly, Figure 10 shows the sample images for targeted fooling attacks. Figure 10a corresponds to the images for the LeNet-5 and Figure 10b present the images for ResNet. In addition to the information described above for Figure 9, Figure 10 also shows the target class. The target class for each image is chosen randomly, however it was made sure that the it is never same as the real class of the image.


Pre-attack: Cat (45.2%) Post-attack: Deer (46.2%) Iterations: 50
Pre-attack: Horse (65.6%) Post-attack: Dog (52.85%) Iterations: 280
Pre-attack: Automobile (93.4%) Post-attack: Aeroplane (48.13%) Iterations: 427
Pre-attack: Bird (72.3%) Post-attack: Truck (51.62%) Iterations: 136
Pre-attack: Bird (46.48%) Post-attack: Aeroplane (60.72%) Iterations: 1

(**a**)


Pre-attack: Automobile (99.99%) Post-attack: Aeroplane (90.36%) Iterations: 41
Pre-attack: Truck (76.08%) Post-attack: Aeroplane (56.06%) Iterations: 1
Pre-attack: Dog (99.58%) Post-attack: Frog (62.15%) Iterations: 312
Pre-attack: Horse (99.63%) Post-attack: Cat (81.18%) Iterations: 136
Pre-attack: Bird (99.99%) Post-attack: Deer (61.8%) Iterations: 156

(**b**)

Figure 9. Samples of single pixel successful untargeted attacks on LeNet-5 and ResNet architectures. Perturbed pixel is marked as red. The pre-attack, post-attack predictions and confidences along with the number of iteration required to fool the CNN are also shown below the images. (**a**) Samples for LeNet-5. (**b**) Samples for ResNet.

To compile the statistical results, we randomly selected a total of 300 images. We then used a 5-fold attacking method, i.e., each image was attacked five times. We also evaluated the performance of the proposed algorithm for the case of 3-pixel and 5-pixel attacks. The distribution of success rate is shown as a histogram and pie chart in Figure 11. Figure 11a shows the success rate distribution for the attacks on LeNet-5 architecture. The left-most end of the histogram represents the images on which all attempts failed. Whereas, the right-most end contains the images with success on all five trials. Although most samples are contained at the edges of the histogram, still there are some samples in the middle. It indicates that the algorithm shows probabilistic behavior, i.e., succeed and failed on the same input image. Additionally, it can be seen that for higher number of pixels, the success rate also increases. It is intuitive since the more the number of perturbed pixels, the greater is the difference between the original image and the new image. Similarly, it can be seen that the success rate for the untargeted attack is higher as compared to the targeted attack.

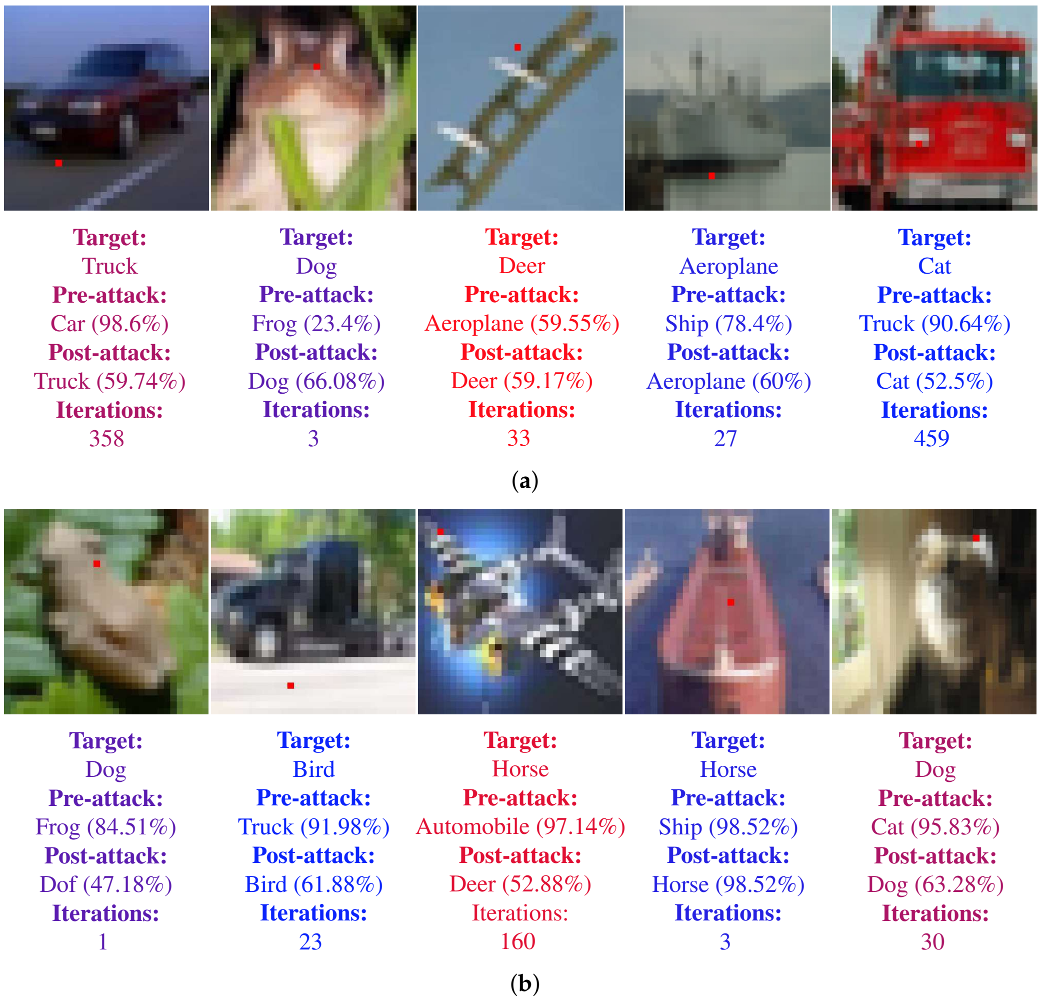

Target: Truck **Pre-attack:** Car (98.6%) **Post-attack:** Truck (59.74%) **Iterations:** 358
Target: Dog **Pre-attack:** Frog (23.4%) **Post-attack:** Dog (66.08%) **Iterations:** 3
Target: Deer **Pre-attack:** Aeroplane (59.55%) **Post-attack:** Deer (59.17%) **Iterations:** 33
Target: Aeroplane **Pre-attack:** Ship (78.4%) **Post-attack:** Aeroplane (60%) **Iterations:** 27
Target: Cat **Pre-attack:** Truck (90.64%) **Post-attack:** Cat (52.5%) **Iterations:** 459
(**a**)
Target: Dog **Pre-attack:** Frog (84.51%) **Post-attack:** Dof (47.18%) **Iterations:** 1
Target: Bird **Pre-attack:** Truck (91.98%) **Post-attack:** Bird (61.88%) **Iterations:** 23
Target: Horse **Pre-attack:** Automobile (97.14%) **Post-attack:** Deer (52.88%) Iterations: 160
Target: Horse **Pre-attack:** Ship (98.52%) **Post-attack:** Horse (98.52%) **Iterations:** 3
Target: Dog **Pre-attack:** Cat (95.83%) **Post-attack:** Dog (63.28%) **Iterations:** 30
(**b**)

Figure 10. Samples of single pixel successful targeted attacks on both LeNet-5 and ResNet architectures. Perturbed pixel is marked as red. The pre-attack, post-attack predictions and confidences along with the number of iteration required to fool the CNN are also shown below the images. (**a**) Samples for LeNet-5. (**b**) Samples for ResNet.

We repeated the same set of experiments for the ResNet. As expected, the attack on ResNet proved to be difficult as compared to LeNet and achieved a lower success rate. Figure 11c summarize the success rate for targeted and untargeted attack on ResNet for the case of 1, 3 and 5-pixel attacks on ResNet. As an overall summary, Figure 11b shows the distribution of the number of success in all the 5-fold attacks. It shows that the proposed algorithm is able to fool the CNNs for a large proportion of images successfully.

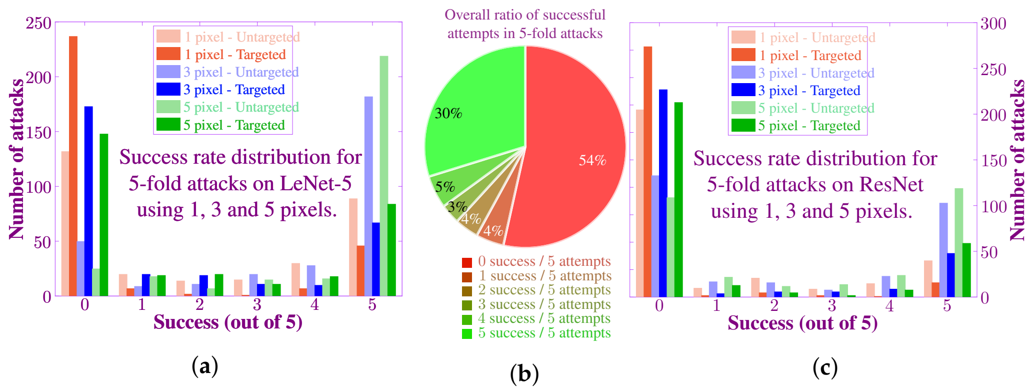


Figure 11. Statistics of the attack success rate for all experimental scenarios. (**a**) The histogram shows the success rate distribution in 5-fold attacking experiments for LeNet-5. Random images from the CIFAR-10 test dataset are attacked five times each using 1,3 and 5-pixels, and the number of successful attempts is recorded as shown in the histogram. The targeted attacks have a lower success rate as compared to untargeted attacks. Additionally, as the number of attacked pixels increases, the success rate also improves. (**b**) The proportion of successful and unsuccessful attacks for all of the experiments. (**c**) Success rate distribution for ResNet in both targeted and untargeted attacks.

5. Conclusions

Experimental results presented in this paper clearly show that, from a computational point of view, the behavior of an rudimentary-intelligent insect, i.e., beetle, is fully capable of fooling artificially intelligent computation models. These results can help provide insight into the nature of artificial intelligent learning systems and their comparison with natural intelligence. Additionally, this paper emphasizes the need for robustness in the designing of artificial learning systems. We demonstrate the efficacy of the proposed CNN attacking algorithm by considering two CNN architectures; LeNet-5 and ResNet, trained on the CIFAR-10 dataset. The statistical results show that the algorithm was able to successfully fool CNN for a large proportion of the input images.

Author Contributions: A.H.K. performed the mathematical and numerical analysis along with the drafting of the manuscript. X.C. proposed the idea and designed the base framework. B.X. contributed to the analysis of the dataset and the drafting of the paper. S.L. contributed to the development of mathematical analysis and verified the numerical analysis. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors used the CIFAR-10 dataset (https://www.cs.toronto.edu/~kriz/cifar.html) and Tensorflow (https://www.tensorflow.org/) to conduct the experiments. Website access date: 1 May 2022.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hilger, K.; Ekman, M.; Fiebach, C.J.; Basten, U. Intelligence is associated with the modular structure of intrinsic brain networks. *Sci. Rep.* **2017**, *7*, 16088. [CrossRef]
2. Genç, E.; Fraenz, C.; Schlüter, C.; Friedrich, P.; Hossiep, R.; Voelkle, M.C.; Ling, J.M.; Güntürkün, O.; Jung, R.E. Diffusion markers of dendritic density and arborization in gray matter predict differences in intelligence. *Nat. Commun.* **2018**, *9*, 1905. [CrossRef] [PubMed]
3. Lee, J.J.; McGue, M.; Iacono, W.G.; Michael, A.M.; Chabris, C.F. The causal influence of brain size on human intelligence: Evidence from within-family phenotypic associations and GWAS modeling. *Intelligence* **2019**, *75*, 48–58. [CrossRef] [PubMed]

4. Gibson, K.R. Evolution of human intelligence: The roles of brain size and mental construction. *Brain Behav. Evol.* **2002**, *59*, 10–20. [CrossRef] [PubMed]
5. Pietschnig, J.; Penke, L.; Wicherts, J.M.; Zeiler, M.; Voracek, M. Meta-analysis of associations between human brain volume and intelligence differences: How strong are they and what do they mean? *Neurosci. Biobehav. Rev.* **2015**, *57*, 411–432. [CrossRef] [PubMed]
6. Mattson, M.P. Superior pattern processing is the essence of the evolved human brain. *Front. Neurosci.* **2014**, *8*, 265. [CrossRef] [PubMed]
7. Jaeger, H. Artificial intelligence: Deep neural reasoning. *Nature* **2016**, *538*, 467. [CrossRef]
8. Vargas, D.V.; Murata, J. Spectrum-diverse neuroevolution with unified neural models. *IEEE Trans. Neural Netw. Learn. Syst.* **2016**, *28*, 1759–1773. [CrossRef]
9. Hassabis, D.; Kumaran, D.; Summerfield, C.; Botvinick, M. Neuroscience-inspired artificial intelligence. *Neuron* **2017**, *95*, 245–258. [CrossRef]
10. Kar, K.; Kubilius, J.; Schmidt, K.; Issa, E.B.; DiCarlo, J.J. Evidence that recurrent circuits are critical to the ventral stream's execution of core object recognition behavior. *Nat. Neurosci.* **2019**, *22*, 974–983. [CrossRef]
11. Bingul, Z. Adaptive genetic algorithms applied to dynamic multiobjective problems. *Appl. Soft Comput.* **2007**, *7*, 791–799. [CrossRef]
12. Sabour, S.; Frosst, N.; Hinton, G.E. Dynamic routing between capsules. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 3856–3866.
13. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
14. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; pp. 1097–1105.
15. Boisseau, R.P.; Vogel, D.; Dussutour, A. Habituation in non-neural organisms: evidence from slime moulds. *Proc. R. Soc. Biol. Sci.* **2016**, *283*, 20160446. [CrossRef] [PubMed]
16. Khan, A.H.; Li, S.; Cao, X. Tracking control of redundant manipulator under active remote center-of-motion constraints: an RNN-based metaheuristic approach. *Sci. China Inf. Sci.* **2021**, *64*, 1–18. [CrossRef]
17. Khan, A.H.; Cao, X.; Li, S.; Luo, C. Using social behavior of beetles to establish a computational model for operational management. *IEEE Trans. Comput. Soc. Syst.* **2020**, *7*, 492–502. [CrossRef]
18. Zivkovic, M.; Bacanin, N.; Venkatachalam, K.; Nayyar, A.; Djordjevic, A.; Strumberger, I.; Al-Turjman, F. COVID-19 cases prediction by using hybrid machine learning and beetle antennae search approach. *Sustain. Cities Soc.* **2021**, *66*, 102669. [CrossRef]
19. Xie, S.; Chu, X.; Zheng, M.; Liu, C. Ship predictive collision avoidance method based on an improved beetle antennae search algorithm. *Ocean Eng.* **2019**, *192*, 106542. [CrossRef]
20. Levi, G.; Hassner, T. Age and gender classification using convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Boston, MA, USA, 7–12 June 2015; pp. 34–42.
21. Svoboda, P.; Hradiš, M.; Maršík, L.; Zemcík, P. CNN for license plate motion deblurring. In Proceedings of the 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, 25–28 September 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 3832–3836.
22. Zhang, Z.; Chen, P.; McGough, M.; Xing, F.; Wang, C.; Bui, M.; Xie, Y.; Sapkota, M.; Cui, L.; Dhillon, J.; et al. Pathologist-level interpretable whole-slide cancer diagnosis with deep learning. *Nat. Mach. Intell.* **2019**, *1*, 236. [CrossRef]
23. zu Belzen, J.U.; Bürgel, T.; Holderbach, S.; Bubeck, F.; Adam, L.; Gandor, C.; Klein, M.; Mathony, J.; Pfuderer, P.; Platz, L.; et al. Leveraging implicit knowledge in neural networks for functional dissection and engineering of proteins. *Nat. Mach. Intell.* **2019**, *1*, 225. [CrossRef]
24. Pasquale, G.; Ciliberto, C.; Odone, F.; Rosasco, L.; Natale, L. Real-world Object Recognition with Off-the-shelf Deep Conv Nets: How Many Objects can iCub Learn? *arXiv* **2015**, arXiv:1504.03154.
25. Ding, Y.; Zhang, W.; Yu, L.; Lu, K. The accuracy and efficiency of GA and PSO optimization schemes on estimating reaction kinetic parameters of biomass pyrolysis. *Energy* **2019**, *176*, 582–588. [CrossRef]
26. Whitley, D. A genetic algorithm tutorial. *Stat. Comput.* **1994**, *4*, 65–85. [CrossRef]
27. Dorigo, M.; Di Caro, G. Ant colony optimization: a new meta-heuristic. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 6–9 July 1999; IEEE: Piscataway, NJ, USA, 1999; Volume 2, pp. 1470–1477.
28. Neshat, M.; Sepidnam, G.; Sargolzaei, M.; Toosi, A.N. Artificial fish swarm algorithm: a survey of the state-of-the-art, hybridization, combinatorial and indicative applications. *Artif. Intell. Rev.* **2014**, *42*, 965–997. [CrossRef]
29. Yang, X.S.; Deb, S. Engineering optimisation by cuckoo search. *arXiv* **2010**, arXiv:1005.2908.
30. Mehrabian, A.R.; Lucas, C. A novel numerical optimization algorithm inspired from weed colonization. *Ecol. Inform.* **2006**, *1*, 355–366. [CrossRef]
31. Nakrani, S.; Tovey, C. On honey bees and dynamic server allocation in internet hosting centers. *Adapt. Behav.* **2004**, *12*, 223–240. [CrossRef]

32. Yang, X.S. Firefly algorithms for multimodal optimization. In Proceedings of the International Symposium on Stochastic Algorithms, Sapporo, Japan, 26–28 October 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 169–178.
33. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [CrossRef]
34. Połap, D.; Woźniak, M. Red fox optimization algorithm. *Expert Syst. Appl.* **2021**, *166*, 114107. [CrossRef]
35. Zhang, M.; Xu, Z.; Lu, X.; Liu, Y.; Xiao, Q.; Taheri, B. An optimal model identification for solid oxide fuel cell based on extreme learning machines optimized by improved Red Fox Optimization algorithm. *Int. J. Hydrogen Energy* **2021**, *46*, 28270–28281. [CrossRef]
36. Hayyolalam, V.; Kazem, A.A.P. Black widow optimization algorithm: a novel meta-heuristic approach for solving engineering optimization problems. *Eng. Appl. Artif. Intell.* **2020**, *87*, 103249. [CrossRef]
37. Hu, G.; Du, B.; Wang, X.; Wei, G. An enhanced black widow optimization algorithm for feature selection. *Knowl. Based Syst.* **2022**, *235*, 107638. [CrossRef]
38. Maqsood, M.; Ghazanfar, M.A.; Mehmood, I.; Hwang, E.; Rho, S. A Meta-Heuristic Optimization Based Less Imperceptible Adversarial Attack on Gait Based Surveillance Systems. *J. Signal Process. Syst.* **2022**, 1–23. [CrossRef]
39. Msika, S.; Quintero, A.; Khomh, F. SIGMA: Strengthening IDS with GAN and Metaheuristics Attacks. *arXiv* **2019**, arXiv:1912.09303.
40. Zang, Y.; Qi, F.; Yang, C.; Liu, Z.; Zhang, M.; Liu, Q.; Sun, M. Word-level textual adversarial attacking as combinatorial optimization. *arXiv* **2019**, arXiv:1910.12196.
41. Su, J.; Vargas, D.V.; Sakurai, K. One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.* **2019**, *23*, 828–841. [CrossRef]
42. Su, J.; Vasconcellos, V.D.; Prasad, S.; Daniele, S.; Feng, Y.; Sakurai, K. Lightweight classification of IoT malware based on image recognition. In Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 23–27 July 2018; IEEE: Piscataway, NJ, USA, 2018; Volume 2, pp. 664–669.
43. Pasquale, G.; Ciliberto, C.; Rosasco, L.; Natale, L. Object identification from few examples by improving the invariance of a deep convolutional neural network. In Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, 9–14 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 4904–4911.
44. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
45. Milletari, F.; Navab, N.; Ahmadi, S.A. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In Proceedings of the 2016 Fourth International Conference on 3D Vision (3DV), Stanford, CA, USA, 25–28 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 565–571.
46. Zhang, K.; Zuo, W.; Gu, S.; Zhang, L. Learning deep CNN denoiser prior for image restoration. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 3929–3938.
47. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. *arXiv* **2013**, arXiv:1312.6199.
48. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and harnessing adversarial examples. *arXiv* **2014**, arXiv:1412.6572.
49. Moosavi-Dezfooli, S.M.; Fawzi, A.; Frossard, P. Deepfool: A simple and accurate method to fool deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2016; pp. 2574–2582.
50. Xu, X.; Chen, X.; Liu, C.; Rohrbach, A.; Darrell, T.; Song, D. Fooling vision and language models despite localization and attention mechanism. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4951–4961.
51. Sun, Y.; Zhang, J.; Li, G.; Wang, Y.; Sun, J.; Jiang, C. Optimized neural network using beetle antennae search for predicting the unconfined compressive strength of jet grouting coalcretes. *Int. J. Numer. Anal. Methods Geomech.* **2019**, *43*, 801–813. [CrossRef]
52. Jiang, X.; Li, S. BAS: Beetle Antennae Search Algorithm for Optimization Problems. *Int. J. Robot. Control* **2018**, *1*, 1–5. [CrossRef]
53. Khan, A.H.; Li, S.; Luo, X. Obstacle avoidance and tracking control of redundant robotic manipulator: An rnn based metaheuristic approach. *IEEE Trans. Ind. Inform.* **2019**, *16*, 4670–4680. [CrossRef]
54. Khan, A.T.; Li, S.; Zhou, X. Trajectory optimization of 5-link biped robot using beetle antennae search. *IEEE Trans. Circuits Syst. II Express Briefs* **2021**, *68*, 3276–3280. [CrossRef]
55. Wu, Q.; Shen, X.; Jin, Y.; Chen, Z.; Li, S.; Khan, A.H.; Chen, D. Intelligent beetle antennae search for UAV sensing and avoidance of obstacles. *Sensors* **2019**, *19*, 1758. [CrossRef] [PubMed]
56. Zhang, J.; Huang, Y.; Ma, G.; Nener, B. Multi-objective beetle antennae search algorithm. *arXiv* **2020**, arXiv:2002.10090.
57. Khan, A.H.; Li, S.; Chen, D.; Liao, L. Tracking Control of Redundant Mobile Manipulator: An RNN based Metaheuristic Approach. *Neurocomputing* **2020**, *400*, 272–284. [CrossRef]
58. Khan, A.T.; Cao, X.; Li, Z.; Li, S. Enhanced beetle antennae search with zeroing neural network for online solution of constrained optimization. *Neurocomputing* **2021**, *447*, 294–306. [CrossRef]
59. Khan, A.T.; Li, S.; Cao, X. Human guided cooperative robotic agents in smart home using beetle antennae search. *Sci. China Inf. Sci.* **2022**, *65*, 1–17. [CrossRef]
60. Ren, Z.; Li, P.; Fang, J.; Li, H.; Chen, Q. SBA: an efficient algorithm for address assignment in ZigBee networks. *Wirel. Pers. Commun.* **2013**, *71*, 719–734. [CrossRef]

61. Khan, A.H.; Cao, X.; Katsikis, V.N.; Stanimirović, P.; Brajević, I.; Li, S.; Kadry, S.; Nam, Y. Optimal Portfolio Management for Engineering Problems Using Nonconvex Cardinality Constraint: A Computing Perspective. *IEEE Access* **2020**, *8*, 57437–57450. [CrossRef]
62. Khan, A.H.; Cao, X.; Li, S.; Katsikis, V.N.; Liao, L. BAS-ADAM: an ADAM based approach to improve the performance of beetle antennae search optimizer. *IEEE/CAA J. Autom. Sin.* **2020**, *7*, 461–471. [CrossRef]
63. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
64. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
65. Krizhevsky, A.; Hinton, G. *Learning Multiple Layers of Features from Tiny Images*; Technical Report; Citeseer: Princeton, NJ, USA, 2009.
66. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. Tensorflow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.