Simple Black-Box Adversarial Attacks on Deep Neural Networks

Nina Narodytska
VMware Research
Palo Alto, USA
n.narodytska@gmail.com

Shiva Kasiviswanathan
Samsung Research America
Mountain View, USA
kasivisw@gmail.com

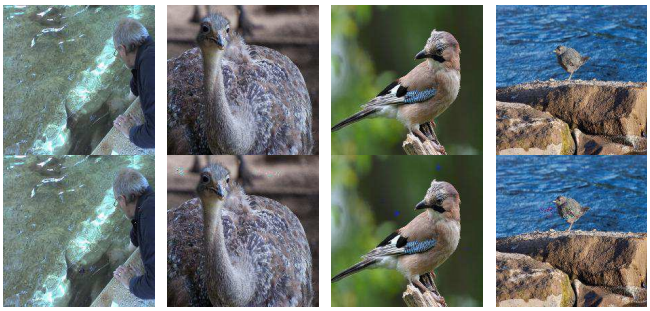
Abstract

Deep neural networks are powerful and popular learning models that achieve state-of-the-art pattern recognition performance on many computer vision, speech, and language processing tasks. However, these networks have also been shown susceptible to crafted adversarial perturbations which force misclassification of the inputs. Adversarial examples enable adversaries to subvert the expected system behavior leading to undesired consequences and could pose a security risk when these systems are deployed in the real world.

In this work, we focus on deep convolutional neural networks and demonstrate that adversaries can easily craft adversarial examples even without any internal knowledge of the target network. Our attacks treat the network as an oracle (black-box) and only assume that the output of the network can be observed on the probed inputs. Our attacks utilize a novel local-search based technique to construct numerical approximation to the network gradient, which is then carefully used to construct a small set of pixels in an image to perturb. We demonstrate how this underlying idea can be adapted to achieve several strong notions of misclassification. The simplicity and effectiveness of our proposed schemes mean that they could serve as a litmus test for designing robust networks.

1. Introduction

Convolutional neural networks (CNNs) are among the most popular techniques employed for computer vision tasks, including but not limited to image recognition, localization, video tracking, and image and video segmentation [8]. Though these deep networks have exhibited good performances for these tasks, they have recently been shown to be particularly susceptible to adversarial perturbations to the input images [26, 9, 18, 21, 20, 13, 10, 29]. Vulnerability of these networks to adversarial attacks can lead to undesirable consequences in many practical applications utilizing these networks. For example, adversarial attacks can be used to subvert fraud detection, malware detection, or mislead autonomous navigation systems [21, 10] and poses a serious security risk (e.g., consider an adversary that can fool an autonomous driving system into not following posted traffic signs). Further strengthening these results is a recent observation by [13] who showed that a significant fraction of adversarial images crafted using the original network are misclassified even when fed to the classifier through a physical world system (such as a camera).



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

Table 1: The top row shows the original images and the bottom row shows the perturbed images. The misclassification is as follows: (a) a stingray misclassified as a sea lion, (b) an ostrich misclassified as a goose, (c) a jay misclassified as a junco, and (d) a water ouzel misclassified as a redshank.

In this paper, we investigate the robustness of state-of-the-art convolutional neural networks (CNNs) with images as inputs to simple black-box adversarial attacks. The rough goal of adversarial attacks in this setting is as follows: Given an image I that is *correctly classified* by a convolutional neural network, construct a transformation of I (say, by adding a small perturbation to some or all the pixels) that now leads to *incorrect classification* by the network. The nature of the incorrectness is based on the adversarial objective. More often than not, in these attacks, the modification done to the image is so subtle that the changes are imperceptible to a human eye. Our proposed attacks also share this property, in addition to being practical and simplistic, thus highlighting a worrying aspect about lack of robustness prevalent in these modern deep learning based vision techniques.

There are two main research directions in the literature on adversarial attacks based on different assumptions about the adversarial knowledge of the target network. The first and the most common line of work assumes that the adversary has detailed knowledge of the network architecture and the parameters resulting from training (or access to the labeled training set) [26, 9, 18, 21]. Using this information, an adversary constructs a perturbation for a given image. The most effective methods are gradient-based: a small perturbation is constructed based on the gradient of the network loss function w.r.t. the input image. Often, adding this small perturbation to the original image leads to a misclassification. In the second line of work an adversary has restricted knowledge about the network from being able to only observe the network's output on some probed inputs [20, 16]. Our work falls into this category. While this *black-box* model is a much more realistic and applicable threat model, it is also more challenging because it considers weak adversaries without knowledge of the network architecture, parameters, or training data. Surprisingly, our results suggest that this level of access and a small number of queries provide sufficient information to construct an adversarial image.

Papernot *et al.* [20] were the first to discuss a black-box attack against deep learning systems. Their attack crucially relies on the observation that there is a *transferability* (*generalization*) property in adversarial examples, i.e., adversarial examples from one model transfers to another. Our proposed attacks on the other hand is much more simple and direct, does not require this transferability property, and hence is more effective in constructing adversarial images, in addition to having some other computational advantages. We demonstrate that our method is capable of constructing adversarial images for several network architectures trained on different datasets. In particular in this paper, we consider the CIFAR10, MNIST, SVHN, STL10, and ImageNet1000 datasets, and two popular network architectures, Network-in-Network [15] and VGG [25]. In Table 1, we show four images from the ImageNet1000 dataset. The original images are in the upper row. The bottom row shows the corresponding perturbed images produced by our algorithm which are misclassified by a VGG CNN-S network [4].

Our Contributions. In this work, we demonstrate the ease of generating adversarial images for modern deep CNNs without knowledge of either the network architecture or its parameters. Our attack strategy is based the idea of *greedy local search*, an iterative search procedure, where in each round a local neighborhood is used to refine the current image and in process optimizing some objective function that depends on the network output. As we operate in a black-box setting, it is not possible to obtain the true gradient of the network loss function, hence we rely on numerical approximations of the gradient. In each round, the local search procedure generates an implicit approximation to the gradient of the network loss function w.r.t. the current image by observing changes in output by changing a few pixels in the current image. This approximate gradient provides a partial understanding of the *influential* pixels in the current image for the output, which is then used to update this image.

We adapt this general strategy to achieve various notions of misclassification with quite small perturbations. With the simplest notion of misclassification, the goal is to alter the input such that the network output is now different from the true class label of the input (e.g., given an image of a cat, alter the image such that the network now fails to identify it as cat). While this is the most commonly used notion of misclassification in the literature [26, 9], many modern computer vision systems (e.g., ImageNet competition entrants) are routinely evaluated based on their top-k predictions. This motivates us to consider a stronger notion of misclassification, that we refer to as *k-misclassification* (Definition 1), where the goal is to alter the input such that the network fails to identify the true label even when relying on top-k predictions (e.g., given an image of a cat, alter the image such that even the top-k predictions of the network does not capture it as cat). We also consider the notion of *targeted misclassification*, where the goal is to take an input and alter it so as to have the network classify it as any chosen target class label that is distinct from the true class label (e.g., given an image of a cat and target class as dog, alter the image such that the network now identifies it as dog).

We perform extensive experimental evaluations on multiple image datasets, and show that our local-search based approach reliably generates adversarial images with little perturbation (even when compared to a recent elegant white-box adversarial attack proposed by Goodfellow *et al.* [9] which needs perfect knowledge of the network). Another feature of our attack is that, by design, our approach only perturbs a very small fraction of the pixels during the adversarial image generation process (e.g., on ImageNet1000 we on average perturb only about 0.5% of the pixels per image). Most previous attacks [26, 9, 18] require the ability to perturb all the pixels in the image. Therefore, interestingly, our results also demonstrate that altering a small fraction of carefully selected pixels suffices to generate adversarial images.

2. Related Work

Starting with the seminal paper by Szegedy *et al.* [26], which showed that the state-of-the-art neural networks are vulnerable to adversarial attacks, there has been significant attention focused on this problem. The research has led to investigation of different adversarial threat models and scenarios [21, 20, 10, 13, 7], computationally efficient attacks [9], perturbation efficient attacks [18], etc.

Szegedy *et al.* [26] used a box-constrained L-BFGS technique to generate adversarial examples. They also showed a transferability (or generalization) property for adversarial

examples, in that adversarial examples generated for one network might also be misclassified by a related network with possibly different hyper-parameters (number of layers, initial weights, etc.). However, the need for a solving a series of costly penalized optimization problems make this technique computationally expensive for generating adversarial examples. This issue was fixed by [9] who motivated by the underlying linearity of the components used to build a network proposed an elegant scheme based on adding perturbation proportional to sign of the network's cost function gradient. Recently, Moosavi *et al.* [18] used an iterative linearization procedure to generate adversarial examples with lesser perturbation. Papernot *et al.* [21] used a notion of *adversarial saliency maps* (based on the saliency maps introduced by [24]) to select the most sensitive input components for perturbation. This attack has been adapted by Grosse *et al.* [10] for generating adversarial samples for neural networks used as malware classifiers. However, all these above described attacks require perfect knowledge of the target network's architecture and parameters which limits their applicability to strong adversaries with the capability of gaining insider knowledge of the target system.

Our focus in this paper is the setting of black-box attacks, where we assume that an adversary has only the ability to use the network as an oracle. The adversary can obtain output from supplied inputs, and use the observed input-output relationship to craft adversarial images.[1] In the context of deep neural networks, a black-box attack was first proposed by Papernot *et al.* [20] with the motivation of constructing an attack on a remotely hosted system.[2] Their general idea is to first approximate the target network by querying it for output labels, which is used to train a substitute network, which is then used to craft adversarial examples for the original network. The success of the attack crucially depends on the transferability property to hold between the original and the substitute network. While empirical evidence exists for the transferability assumption, it usually results in a degradation in the effectiveness of the attacks, and in some cases this degradation can be upwards of 30% [20]. A very recent result by Liu *et al.* [16] has also highlighted that this assumption should be treated carefully. Our black-box attack is more direct, and completely avoids the transferability assumption, making it far more applicable. We also avoid the overhead of gathering data and training a substitute network.

A complementary line of work has focused on building defenses against adversarial attacks. Although designing defenses is beyond scope of this paper, it is possible that adapting the previous suggested defense solutions such as *Jacobian-based regularization* [11] and *distillation* [22] can reduce the efficacy of our proposed attacks. Moreover, the recently proposed technique of *differentially private training* [1] can also prove beneficial here.

The study of adversarial instability have led to development of solutions that seeks to improve training to in return increase the robustness and classification performance of the network. In some case, adding adversarial examples to the training (*adversarial training*) set can act like a regularizer [26, 9, 18]. The phenomenon of adversarial instability has also been theoretically investigated for certain families of classifiers under various models of (semi) random noise [6, 7]. However, due to peculiar nature of adversarial images generated by our approaches, a simple adversarial training is only mildly effective in preventing future similar adversarial attacks. The security of machine learning in settings distinct from deep neural networks is also an area of active research with various known attacks under different threat models [27, 19]. We refer the reader to a recent survey by McDaniel *et al.* [17] for a review of developments there.

3. Preliminaries

Notation and Normalization. We denote by $[n]$ the set $\{1, \ldots, n\}$. The dataset of images is partitioned into train and test (or validation) subsets. An element of a dataset is a pair $(I, c(I))$ for an image I and a ground truth label $c(I)$ of this image. We assume that the class labels are drawn from the set $\{1, \ldots, C\}$, i.e., we have a set of $C \in \mathbb{N}$ possible labels. We assume that images have ℓ channels (in experiments we use the RGB format) and are of width $w \in \mathbb{N}$ and height $h \in \mathbb{N}$. We say that (b, x, y) is a coordinate of an image for channel b and location (x, y), and $(\star, x, y)$ is a pixel of an image where $(\star, x, y)$ represents all the ℓ coordinates corresponding to different channels at location (x, y). $I(b, x, y) \in \mathbb{R}$ is the value of I at the (b, x, y) coordinate, and similarly $I(\star, x, y) \in \mathbb{R}^\ell$ represents the vector of values of I at the $(\star, x, y)$ pixel.

It is a common practice to normalize the image before passing it to the network. Note that a normalized image have the same dimensions as the original image, but differs in the coordinate values. Since the normalization procedures are generally standard, we assume that the adversary can also carry them out. As we always work with normalized images, in the following, a reference to image means a normalized input image. We denote by LB and UB two constants such that all the coordinates of all the normalized images fall in the range $[\mathrm{LB}, \mathrm{UB}]$. Generally, $\mathrm{LB} < 0$ and $\mathrm{UB} > 0$. We denote by $\mathbb{I} \subset \mathbb{R}^{\ell \times w \times h}$ the space of all (valid) images which satisfy the following property: for every $I \in \mathbb{I}$, for all coordinates $(b, x, y) \in [\ell] \times [w] \times [h]$, $I(b, x, y) \in [\mathrm{LB}, \mathrm{UB}]$.

Network Input and Output. We denote by NN a trained convolutional neural network (trained on some set of training images). NN takes an image I as an input and outputs a

[1] These kind of attacks are also known as *differential attacks* motivated by the use of the term in *differential cryptanalysis* [3].

[2] Papernot *et al.* [19] have recently extended this attack beyond deep neural networks to other classes of machine learning techniques.

vector $\text{NN}(I) = (o_1, \ldots, o_C)$, where o_j denotes the probability as determined by NN that image I belongs to class j. The top prediction of NN on I is the class label with the largest probability score in $\text{NN}(I)$. Similarly, the top-k (for $k \in [C]$) predictions are obtained by taking the top-k class labels by decreasing probability scores (with ties broken arbitrarily). We denote $\pi(\text{NN}(I), k)$ a function that returns a set of top-k class labels. For example, if $\text{NN}(I) = (0.25, 0.1, 0.2, 0.45)$, then $\pi(\text{NN}(I), 1) = \{4\}$ (corresponding to the location of the entry 0.45), $\pi(\text{NN}(I), 2) = \{4, 1\}$, etc. Our adversarial approaches do not require access to the complete probability vector ($\text{NN}(I)$), but just access to the probability score for the class label of interest (which depends on the notion of mis-classification used) and the π vector (used for early stopping). This is slightly different from the adversary presented in [20] that requires access to the class label assigned by the network.

Misclassification Notions. First, we define misclassification for a NN. We use two different notions of misclassification [21]. The first one, referred to as k-misclassification for $k \in [C]$, is defined as follows.

Definition 1 (k-misclassification) *A neural network* NN k-misclassifies *an image I with true label $c(I)$ iff the output* $\text{NN}(I)$ *of the network satisfies* $c(I) \notin \pi(\text{NN}(I), k)$.

In other words, k-misclassification means that the network ranks the true label below at least k other labels. Traditionally the literature on adversarial attacks have only considered the case where $k = 1$. Note that an adversary that achieves a k-misclassification for $k > 1$ is a stronger adversary than one achieving an 1-misclassification (k-misclassification implies k'-misclassification for all $1 \leq k' \leq k$). To the best of our knowledge, ours is the first result about adversarial attacks on deep neural networks achieving k-misclassification for $k > 1$.

We also provide adversarial attacks for a related but distinct notion of targeted misclassification that was first considered in the context of deep neural networks by Papernot *et al.* [21]. Given a target class label T, we say that a neural network *targeted misclassifies* an image I with true label $c(I) \neq T$ iff the output $T \in \pi(\text{NN}(I), 1)$. Note that in general, the targeted and k-misclassification (for $k > 1$) notions are irreducible to one other, so one of them is not necessarily stronger than the other. The flexibility of our approach allows us to achieve either notion based on the requirements.

Adversarial Goal. In our setting, an adversary ADV is a function that takes in image I as input and whose output is another image $\text{ADV}(I)$ (with same number of coordinates as I). We define an adversarial image as one that *fools* a network into misclassification (under one of the above notions). The goal of adversarial attacks is to design this function ADV that succeeds in fooling the network for a large set of images. Ideally, we would like to achieve this misclassification[3] by adding only some small perturbation (under some distance metric) to the image.

4. Adversarial Image Generation

In this section, we present an overview of our general adversarial attack strategy that is based on performing a greedy local search over the image space. Note that unlike some of the previous adversarial attacks [26, 9, 18, 21, 10], our threat model does not assume access to the true network gradient factors, making any gradient (or Jacobian based) methods not directly applicable. Instead, our attacks use a local search technique to construct an implicit approximation to the network gradient which is then used to guide the generation of the perturbed image.

Local search procedure, is an incomplete search procedure that is widely used for solving combinatorial problems appearing in diverse domains such as graph clustering, scheduling, logistics, and verification [14]. For a general optimization problem it works as follows. Consider an objective function $f(\mathbf{z}) : \mathbb{R}^n \to \mathbb{R}$ where the goal is to minimize $f(\mathbf{z})$. The local-search procedure works in rounds, where each round consists of two steps. Let $\mathbf{z}_{i-1}$ be the solution iterate after round $i-1$. Consider round i. The first step is to select a small subset of points $Z = \{\hat{\mathbf{z}}_1, \ldots, \hat{\mathbf{z}}_n\}$, a so called *local neighborhood*, and evaluate $f(\hat{\mathbf{z}}_j)$ for every $\hat{\mathbf{z}}_j \in Z$. Usually, the set Z consist of points that are close to current $\mathbf{z}_{i-1}$ for some measure of distance which is domain specific. The second step selects a new solution $\mathbf{z}_i$ taking into account the previous solution $\mathbf{z}_{i-1}$ and the points in Z. Hence, $\mathbf{z}_i = g(f(\mathbf{z}_{i-1}), f(\hat{\mathbf{z}}_1), \ldots, f(\hat{\mathbf{z}}_n))$, where g is some pre-defined *transformation function*.

Now an image I can be perturbed in multiple ways. In this paper, we utilize a simple class of sign-preserving perturbation functions defined as follows.[4] Let $\text{PERT}(I, p, x, y)$ be a function that takes as input an image I, a perturbation parameter $p \in \mathbb{R}$, and a location (x, y) in the image, and outputs an image $I_p^{(x,y)} \in \mathbb{R}^{\ell \times w \times h}$, defined as:

$$I_p^{(x,y)}(b, u, v) \stackrel{\text{defn}}{=} \begin{cases} (I(b, u, v) \text{ if } x \neq u \text{ or } y \neq v \\ p \times \text{sign}(I(b, u, v)) \text{ otherwise} \end{cases} \quad (1)$$

In other words, the image $I_p^{(x,y)} = \text{PERT}(I, p, x, y)$ has same values as image I at all pixels except the pixel $(\star, x, y)$.

We first describe our local-search based attack for achieving k-misclassification (Definition 1) where an adversarial attack ensures that the true label does not appear in the top-k predictions of the network. The attack for achieving targeted misclassification is quite similar and we discuss that later.

[3] Note that the misclassification is at test time, once the trained network has been deployed.

[4] The use of this specific perturbation function is not very important for our attack scheme and we use it for its simplicity.

Adversarial Attack for k-misclassification. We set up a local search procedure as follows. Our optimization problem will try to minimize the probability that the network determines an perturbed image has the class label of the original image, and by using a local-search procedure we generate perturbed images which differ from the original image in only few pixels. Intuitively, in each round, our local-search procedure computes an implicit approximation to the gradient of the current image by understanding the influence of a few pixels on the output, which is used to update this image.

First, we need to define the cost function f. Let I be the image (with true label $c(I)$) whose adversarial image we want to generate for a target neural network NN. For some input image $\hat{I}$, we use the objective function $f_{c(I)}(\hat{I})$ which equals the probability assigned by the network NN that $\hat{I}$ belongs to class $c(I)$. More formally, $f_{c(I)}(\hat{I}) = o_{c(I)}$ where $\text{NN}(\hat{I}) = (o_1, \ldots, o_C)$, with o_j denoting the probability as determined by NN that image $\hat{I}$ belongs to class j. Our local search aims to minimize this function.

Second, we consider how to form a neighborhood set of images. As mentioned above, the local-search procedure operates in rounds. Let $\hat{I}_{i-1}$ be the image after round $i-1$. Our neighborhood will consist of images that are different in one pixel from the image $\hat{I}_{i-1}$. In other words, if we measure the distance between $\hat{I}_{i-1}$ and any image in the neighborhood as the number of perturbed pixels, then this distance is the same (equal to one) for all of them. Therefore, we can define the neighborhood in terms of a set of pixel locations. Let $(P_X, P_Y)_i$ be a set of pixel locations. For the first round $(P_X, P_Y)_0$ is randomly generated. At each subsequent round, it is formed based on a set of pixel locations which were perturbed in the previoqus round. Let $(P_X^*, P_Y^*)_{i-1}$ denote the pixel locations that were perturbed in round $i-1$ (formally defined below). Then

$$(P_X, P_Y)_i = \bigcup_{\{(a,b)\in(P_X^*,P_Y^*)_{i-1}\}} \bigcup_{\substack{\{x\in[a-d,a+d],\\ y\in[b-d,b+d]\}}} (x,y),$$

where d is a parameter. In other words, we consider pixels that were perturbed in the previous round, and for each such pixel we consider all pixels in a small square with the side length $2d$ centered at that pixel. This defines the neighborhood considered in round i.

Third, we describe the transformation function g of a set of pixel locations. The function g takes as input an image $\hat{I}$, a set of pixel locations (P_X, P_Y), a parameter t that defines how many pixels will be perturbed by g, and two perturbation parameters p and r. In round i of the local-search procedure, $\mathcal{I} = \bigcup_{(x,y)\in(P_X,P_Y)_{i-1}}\{\textsc{Pert}(\hat{I}_{i-1}, p, (x,y))\}$, where PERT is the perturbation function defined through (1). Then it computes the score of each image in $\mathcal{I}$ as $\forall \tilde{I} \in \mathcal{I} : \text{score}(\tilde{I}) = f_{c(I)}(\tilde{I})$, and it sorts (in decreasing order) images in $\mathcal{I}$ based on the above score function to construct $\text{sorted}(\mathcal{I})$. Pixels whose perturbation lead to a larger decrease of f are more likely useful in constructing an adversarial candidate. From $\text{sorted}(\mathcal{I})$, it records a set of pixel locations $(P_X^*, P_Y^*)_i$ based on the first t elements of $\text{sorted}(\mathcal{I})$, where the parameter t regulates the number of pixels perturbed in each round. Formally, $(P_X^*, P_Y^*)_i =$

$$\{(x,y) : \textsc{Pert}(\hat{I}_{i-1}, p, (x,y)) \in \text{sorted}(\mathcal{I})[1:t]\},$$

where $\text{sorted}(\mathcal{I})[1:t]$ represents the first t sorted images in $\text{sorted}(\mathcal{I})$. Finally, $\hat{I}_i$ is constructed from $\hat{I}_{i-1}$ by perturbing each pixel in location $(x,y) \in (P_X^*, P_Y^*)_i$ with a perturbation value r. The perturbation is performed using a simple cyclic rounding procedure (CYCLIC) so that we make sure that all coordinate values in $\hat{I}_i$ are within the valid bounds of LB and UB. The cyclic rounding provides pixel values that are closer (in absolute sense) to their original values than a simple rounding scheme. Note that at the end of every round i, $\hat{I}_i$ is a valid image from the original image space $\mathbb{I}$.

We want to point out that the function g uses two perturbation parameters, p and r. The value of r is kept small in the range $[0, 2]$. On the other hand, we do not put any explicit restrictions on the value of p. The best choice of p will be one that facilitates the identification of the "best" pixels to perturb in each round. In our experiments, we adjust the value of p automatically during the search. We defer this discussion to the experimental section.

Algorithm LOCSEARCHADV presents the complete pseudocode of our local-search procedure. At a high level, the algorithm takes an image as input, and in each round, finds some pixel locations to perturb using the above defined objective function and then applies the above defined transformation function to these selected pixels to construct a new (perturbed) image. It terminates if it succeeds to push the true label below the kth place in the confidence score vector at any round. Otherwise, it proceeds to the next round (for a maximum of R rounds). Note that the number of pixels in an image perturbed by Algorithm LOCSEARCHADV is at most $t \times R$ and in practice (see Tables 2, 3, and 4 in Section 5) it is much less. This is in stark contrast with most existing adversarial attack schemes [26, 9, 18] that operate by applying the same perturbation on each individual pixel.

In Section 5, we demonstrate the efficacy of Algorithm LOCSEARCHADV in constructing adversarial images.

Adversarial Attack for Targeted Misclassification. It is straightforward to change Algorithm LOCSEARCHADV to achieve targeted misclassification, where we want a network to (incorrectly) have its top prediction as a given target label $T \in [C]$ and $T \neq c(I)$. In fact, we only need to change the cost function in LOCSEARCHADV, so that we *maximize* the probability that an image I belongs to target class. Namely, we define a cost function $f_{c(I)}(\tilde{I}) = o_T$ with $\text{NN}(\tilde{I}) = (o_1, \ldots, o_C)$ and we now sort the generated scores in an

Algorithm 1 CYCLIC (r, b, x, y)

Assumptions: Perturbation parameter $r \in [0, 2]$ and $\text{LB} \leq 0 \leq \text{UB}$
Output: Perturbed image value at the coordinate (b, x, y) are in $[\text{LB}, \text{UB}]$

if $rI(b, x, y) < \text{LB}$ **then**
  return $rI(b, x, y) + (\text{UB} - \text{LB})$
else if $rI(b, x, y) > \text{UB}$ **then**
  return $rI(b, x, y) - (\text{UB} - \text{LB})$
else
  return $rI(b, x, y)$
end if

Algorithm 2 LOCSEARCHADV (NN)

Input: Image I with true label $c(I) \in \{1, \ldots, C\}$, two perturbation parameters $p \in \mathbb{R}$ and $r \in [0, 2]$, and four other parameters: the half side length of the neighborhood square $d \in \mathbb{N}$, the number of pixels perturbed at each round $t \in \mathbb{N}$, the threshold $k \in \mathbb{N}$ for k-misclassification, and an upper bound on the number of rounds $R \in \mathbb{N}$.
Output: Success/Failure depending on whether the algorithm finds an adversarial image or not

$\hat{I}_0 = I, i = 1$
Pick 10% of pixel locations from I at random to form $(P_X, P_Y)_0$
while $i \leq R$ **do**
{Computing the function g using the neighborhood}
  $\mathcal{I} \leftarrow \bigcup_{(x,y)\in(P_X,P_Y)_{i-1}} \{\text{PERT}(\hat{I}_{i-1}, p, x, y)\}$
  Compute $\text{score}(\tilde{I}) = f_{c(I)}(\tilde{I})$ for each $\tilde{I} \in \mathcal{I}$
  $\text{sorted}(\mathcal{I}) \leftarrow$ images in $\mathcal{I}$ sorted by descending order of score
  $(P^*_X, P^*_Y)_i \leftarrow \{(x, y) : \text{PERT}(\hat{I}_{i-1}, p, x, y) \in \text{sorted}(\mathcal{I})[1 : t]\}$ (with ties broken arbitrarily)
{Generation of the perturbed image $\hat{I}_i$}
  for $(x, y) \in (P^*_X, P^*_Y)_i$ and each channel b **do**
    $\hat{I}_i(b, x, y) \leftarrow \text{CYCLIC}\ (r, b, x, y)$
  end for
{Check whether the perturbed image $\hat{I}_i$ is an adversarial image}
  if $c(I) \notin \pi(\text{NN}(\hat{I}_i), k)$ **then**
    return Success
  end if
{Update a neighborhood of pixel locations for the next round}
  $(P_X, P_Y)_i \leftarrow \bigcup_{\{(a,b)\in(P^*_X,P^*_Y)_{i-1}\}} \bigcup_{\substack{\{x\in[a-d,a+d],\\ y\in[b-d,b+d]\}}} (x, y)$
  $i \leftarrow i + 1$
end while
return Failure

increasing order. The remaining pieces of the local search procedure remains as in Algorithm LOCSEARCHADV.

5. Experimental Evaluation

We start by describing our experimental setup. We used Caffe and Torch machine learning frameworks to train the networks. All algorithms to generate adversarial images were implemented in Lua within Torch 7. All experiments were performed on a cluster of GPUs using a single GPU for each run. We use 5 popular datasets: MNIST, CIFAR10, SVHN, STL10, and ImageNet1000. We trained Network-in-Network [15] and VGG [25] for MNIST, CIFAR, SVHN, STL10, with minor adjustments for the corresponding image sizes. Network-in-Network is a building block of the commonly used GoogLeNet architecture that has demonstrated very good performance on medium size datasets, e.g. CIFAR10 [28]. VGG is another powerful network that proved to be useful in many applications beyond image classification, like object localization [23]. We trained each model in two variants: with and without batch normalization [12]. Batch normalization was placed before a ReLU layer in all networks. For the ImageNet1000 dataset, we used pre-trained VGG models from [5]. All Caffe VGG models were converted to Torch models using the *loadcaffe* package [30].

We use the standard top-k error metric for evaluating the classification performance of a network. Tables 2 and 3 (the second column ERRTOP-1) show the top-1 (base) error for all datasets and models that we considered. The results are comparable with the known state-of-the-art results on these datasets [2].

Related Techniques. There are quite a few approaches for generating adversarial images (as discussed in Section 2). Most of these approaches require access to the network architecture and its parameter values [26, 9, 18, 21], making them not entirely suitable for a direct comparison with our black-box approach. The general idea behind these previous white-box attacks is based on the evaluating the network's sensitivity to the input components in order to determine a perturbation that achieves the adversarial misclassification goal. Among these approaches, the white-box attack approach (known as the "fast-gradient sign method", FGSM for short) suggested by [9] stands out for being able to efficiently generate adversarial images. Here we compare the performance of our proposed black-box attack against FGSM. Without general guidelines for setting ϵ for FGSM, we experimented with several values of ϵ starting from 0.07 and increasing this number. We found that the value $\epsilon = 0.2$[5] was the smallest value where the fast-gradient sign method started to yield competitive performance compared to our algorithm. Smaller values of ϵ leads to generation of fewer adversarial images, e.g., at $\epsilon = 0.1$, the percentage of generated adversarial images is reduced by around 10% as compared to the value at $\epsilon = 0.2$ for CIFAR10 on the NinN model (similar on other datatsets). Larger values of ϵ tends to generate more adversarial images, but this comes at the cost of an increase in the perturbation.

Implementing Algorithm LOCSEARCHADV. For each image I, we ran Algorithm LOCSEARCHADV (LSA, for short) for at most 150 rounds, perturbing 5 pixels at each round, and use squares of side length 10 to form the neighborhood (i.e., $R = 150, t = 5, d = 5$). With this setting of parameters, we perturb a maximum of $t \times R = 750$ pixels in an image. The perturbation parameter p was adaptively adjusted during the search. Though not critical, doing so helps in faster determination of the most helpful pixels in generating the adversarial image. Let I be the original image. For some round i of the algorithm, define $\bar{o}_{c(I)} = \text{avg}_{(x,y)}\{o_{c(I)} : (x, y) \in (P^*_X, P^*_Y)_{i-1}\}$, where $o_{c(I)}$ is the probability assigned to class label $c(I)$

[5] For the ImageNet1000 dataset, we set ϵ differently as discussed later.

Dataset	ErrTop-1	ErrTop-1 (Adv)	CONF	PTB	#PTBPIXELS (%)	TIME(in sec)	Technique	Network
NNs trained with batch normalization								
CIFAR10	11.65	97.63	0.47	0.04	3.75	0.68	LSA (Ours)	NinN
CIFAR10		70.69	0.55	0.20	100.00	0.01	FGSM [9]	NinN
CIFAR10	11.62	97.51	0.74	0.04	3.16	0.78	LSA (Ours)	VGG
CIFAR10		11.62	–	–	–	–	FGSM [9]	VGG
STL10	29.81	58.17	0.42	0.02	1.20	7.15	LSA (Ours)	NinN
STL10		54.85	0.53	0.20	100.00	0.03	FGSM [9]	NinN
STL10	26.50	65.76	0.47	0.02	1.11	13.90	LSA (Ours)	VGG
STL10		26.50	–	–	–	–	FGSM [9]	VGG
SVHN	9.71	97.06	0.47	0.05	4.51	1.02	LSA (Ours)	NinN
SVHN		48.62	0.49	0.20	100.00	0.02	FGSM [9]	NinN
SVHN	4.77	81.10	0.66	0.07	5.43	2.15	LSA (Ours)	VGG
SVHN		4.77	–	–	–	–	FGSM [9]	VGG
MNIST	0.33	91.42	0.54	0.20	2.24	0.64	LSA (Ours)	NinN
MNIST		1.65	0.58	0.20	100.00	0.02	FGSM [9]	NinN
MNIST	0.44	93.48	0.63	0.21	2.20	0.64	LSA (Ours)	VGG
MNIST		0.44	–	–	–	–	FGSM [9]	VGG
NNs trained without batch normalization								
CIFAR10	16.54	97.89	0.72	0.04	3.24	0.58	LSA (Ours)	NinN
CIFAR10		93.67	0.93	0.20	100.00	0.02	FGSM [9]	NinN
CIFAR10	19.79	97.98	0.77	0.04	2.99	0.72	LSA (Ours)	VGG
CIFAR10		90.93	0.90	0.20	100.00	0.04	FGSM [9]	VGG
STL10	35.47	52.65	0.56	0.02	1.17	6.42	LSA (Ours)	NinN
STL10		87.16	0.94	0.20	100.00	0.04	FGSM [9]	NinN
STL10	43.91	59.38	0.52	0.01	1.09	19.65	LSA (Ours)	VGG
STL10		91.36	0.93	0.20	100.00	0.10	FGSM [9]	VGG
SVHN	6.15	92.31	0.68	0.05	4.34	1.06	LSA (Ours)	NinN
SVHN		73.97	0.84	0.20	100.00	0.01	FGSM [9]	NinN
SVHN	7.31	88.34	0.68	0.05	4.09	1.00	LSA (Ours)	VGG
SVHN		76.78	0.89	0.20	100.00	0.04	FGSM [9]	VGG

Table 2: Results for four datasets: CIFAR10, STL10, SVHN, and MNIST. The entries denote by denoted by "– " are the cases where FGSM fails to produce any adversarial image in our experimental setup.

in $\text{NN}(\text{PERT}(\hat{I}_{i-1}, p, x, y))$ (here $\bar{o}_{c(I)}$ provides an approximation of the average confidence of the network NN in predicting the true label over perturbed images). At each round, we increase the value of p if $\bar{o}_{c(I)}$ is close to one and decrease p if $\bar{o}_{c(I)}$ is low, e.g., below 0.3. To avoid perturbing the most sensitive pixels frequently, we make sure that if a pixel is perturbed in a round then we exclude it from consideration for the next 30 rounds.

Results for 1-misclassification. For ease of comparison with FGSM [9], we set $k = 1$ and focus on achieving 1-misclassification.Tables 2 and 3 show the results of our experiments on the test sets. The first column shows the dataset name. The second column (ERRTOP-1) presents the top-1 misclassification rate on the corresponding test dataset without any perturbation (base error). ERRTOP-1 (ADV) is the top-1 misclassification rate where each original image in the test set was replaced with an generated perturbed image (using either our approach or the fast-gradient sign method [9] which is denoted as FGSM).[6]

In the following, we say an adversarial generation technique ADV, given an input image I, *succeeds* in generating an adversarial image $\text{ADV}(I)$ for a network NN iff $c(I) \in \pi(\text{NN}(I), 1)$ and $c(I) \notin \pi(\text{NN}(\text{ADV}(I)), 1)$. The CONF column shows the average confidence over all successful adversarial images for the corresponding technique. The PTB represents the mean absolute error between the image and its adversarial counterpart averaged over successful adversarial images. More formally, let $\mathcal{T}$ denote the test set and $\mathcal{T}_{Adv} \subseteq \mathcal{T}$ denote the set of images in $\mathcal{T}$ on which ADV is successful. Then, PTB is defined as:

$$\frac{1}{|\mathcal{T}_{Adv}|} \sum_{I \in \mathcal{T}_{Adv}} \frac{1}{\ell \cdot w \cdot h} \sum_{b,x,y} |I(b,x,y) - \text{ADV}(I)(b,x,y)|,$$

where $I \in \mathbb{R}^{\ell \times w \times h}$ is the original image and $\text{ADV}(I) \in \mathbb{R}^{\ell \times w \times h}$ is the corresponding adversarial image. Note that the inner summation is measuring the mean absolute error between I and $\text{ADV}(I)$. The #PTBPIXELS column shows the average percentage of perturbed pixels in the successful adversarial images. For Algorithm LOCSEARCHADV the number of pixels perturbed also provides a bound on the average number of network evaluations (oracle queries) used. Similarly, TIME column shows the average time (in seconds) to generate a successful adversarial image. The last column indicates the type of network architecture.

As is quite evident from these results, Algorithm LOCSEARCHADV is more effective than the fast-gradient sign method in generating adversarial images, even without having access to the network architecture and its parameter values. The difference is quite prominent for networks trained with batch normalization as here we noticed that the fast-gradient sign method has difficulties producing adversarial images. In general, we observed that models trained with batch normalization are somewhat more resilient to adversarial perturbations probably because of the regularization properties of batch normalization [12]. We are not aware of any previous results in the adversarial image generation literature that have factored in the effects of batch normalization.

Another advantage with our approach is that it modifies a very tiny fraction of pixels as compared to all the pixels perturbed by FGSM, and also in many cases with far less average perturbation. Putting these points together demonstrates that Algorithm LOCSEARCHADV is successful in generating more adversarial images than FGSM, while modifying far fewer pixels and adding less noise per image. On the other side, FGSM takes lesser time in the generation process and generally seems to produce higher confidence scores for the adversarial (misclassified) images.

Table 3 shows the results for several variants of VGG network trained on the ImageNet1000 dataset. These networks do not have batch normalization layers [5, 30]. We set $\epsilon = 1$ for the fast-gradient sign method as a different pre-processing technique was used for this network (we converted these networks from pre-trained Caffe models).

[6] Note that by explicitly constraining the number of pixels that can be perturbed, as we do in our approach, it might be impossible to get to a 100% misclassification rate on some datasets. Similarly, FGSM fails to achieve a 100% misclassification rate even with larger values of ϵ [18].

ErrTop-1	ErrTop-1 (Adv)	conf	ptb	#ptbpixels (%)	time (in sec)	Technique	Network
58.27	93.59	0.29	0.29	0.43	12.72	LSA (Ours)	VGG CNN-S (Caffe)
	85.51	0.49	1.00	100.00	4.74	FGSM [9]	VGG CNN-S (Caffe)
58.96	91.36	0.28	0.29	0.40	10.01	LSA (Ours)	VGG CNN-M (Caffe)
	87.85	0.48	1.00	100.00	4.36	FGSM [9]	VGG CNN-M (Caffe)
58.80	92.82	0.29	0.30	0.41	11.09	LSA (Ours)	VGG CNN-M 2048 (Caffe)
	88.43	0.52	1.00	100.00	4.42	FGSM [9]	VGG CNN-M 2048 (Caffe)
46.40	72.07	0.30	0.54	0.55	73.64	LSA (Ours)	VGG ILSVRC 19 (Caffe)
	85.05	0.52	1.00	100.00	23.94	FGSM [9]	VGG ILSVRC 19 (Caffe)

Table 3: Results for the ImageNet1000 dataset using a center crop of size 224×224 for each image.

k	ErrTop-k	ErrTop-k (Adv)	conf	ptb	#ptbpixels (%)	time (in sec)	Network
1	16.54	97.89	0.72	0.04	3.24	0.58	NinN
2	6.88	80.81	0.89	0.07	6.27	1.51	NinN
3	3.58	70.23	0.90	0.08	6.78	1.72	NinN
4	1.84	60.00	0.90	0.08	7.27	1.92	NinN

Table 4: Results for k-misclassification using Algorithm LocSearchAdv for CIFAR10.

Target	Target Classif. (Adv) %	conf	ptb	#ptbpixels (%)	time (in sec)	Network
airplane	70.78	0.67	0.06	6.32	0.33	NinN
automobile	60.56	0.66	0.08	6.70	0.31	NinN
bird	71.60	0.70	0.06	6.14	0.34	NinN
cat	36.60	0.74	0.06	4.84	0.16	NinN
deer	29.48	0.73	0.05	4.86	0.14	NinN
dog	30.11	0.78	0.02	2.57	0.10	NinN
frog	20.91	0.77	0.06	4.62	0.09	NinN
horse	27.40	0.74	0.05	4.32	0.11	NinN
ship	23.34	0.78	0.06	4.09	0.08	NinN
truck	32.61	0.75	0.08	5.39	0.14	NinN

Table 5: Results for targeted misclassification using lusing Algorithm LocSearchAdv for CIFAR10.

Results are similar to that observed on the smaller datasets. In most cases, our proposed local-search based approach is more successful in generating adversarial images while on average perturbing less than 0.55% of the pixels.

Results for k-misclassification ($k > 1$). We now consider achieving k-misclassification for $k > 1$ using LocSearchAdv. Table 4 shows the results as we change the goal from 1-misclassification to 4-misclassification on CIFAR10. We use the same parameters as before for LocSearchAdv. As one would expect, as we increase the value of k, the effectiveness of the attack decreases, perturbation and time needed increases. But overall our local-search procedure is still able to generate a large fraction of adversarial images at even $k = 4$ with a small perturbation and computation time, meaning that these images will fool even a system that is evaluated on a top-4 classification criteria. We are not aware of a straightforward extension of the fast-gradient sign method [9] to achieve k-misclassification.

Results for Targeted Misclassification. Table 5 shows the results for achieving targeted misclassification using LocSearchAdv on CIFAR10. We choose each of the 10 labels (first column) as individual targets, and the second column shows the percentage[7] of images in the dataset that when altered subsequently leads the network to classify it as the target label. The results show that for each target class a large number of images can be perturbed with little noise to get targeted misclassification, and some classes (such as 'airplane') are particularly amenable to these attacks.

Even Weaker Adversarial Models. We also consider a weaker model where the adversary does not even have a black-box (oracle) access to the network (NN) of interest, and has to rely on a black-box access to somewhat of a "similar" (proxy) network as NN. For example, the adversary might want to evade a spam filter A, but might have to develop adversarial images by utilizing the output of a spam filter B, which might share properties similar to A.

We trained several modifications of NinN model for CIFAR10, varying the initial value of the learning rate, the size of filters, and the number of layers in the network. We observed that between 25% to 43% of adversarial 1-misclassified images generated by Algorithm LocSearchAdv using the original network were also 1-misclassified by these modified networks. This observation demonstrates the wider applicability of our attack scheme.

6. Conclusion

We investigate the inherent vulnerabilities in modern CNNs to practical black-box adversarial attacks. We present approaches that can efficiently locate a small set of pixels, without knowing any parameter information about the network, which when perturbed lead to misclassification by a deep neural network. Our extensive experimental results, somewhat surprisingly, demonstrates the effectiveness of our simple approaches in generating adversarial examples.

Defenses against these attacks is an interesting research direction. However, we note that here that by limiting the perturbation to some pixels (being localized) the adversarial images generated by our local-search based approach do not represent the distribution of the original data. This means for these adversarial images, the use of adversarial training, a technique of training (or fine-tuning) networks on adversarial images to build more robust classifiers, is not very effective. In fact, even with adversarial training we noticed that the networks ability to resist new local-search based adversarial attack improves only marginally (on average between 1-2%). On the other hand, we suspect that one possible countermeasure to these localized adversarial attacks could be based on performing a careful analysis of the oracle queries to thwart the attempts to generate an adversarial image.

Finally, we believe that our local-search approach can also be used for attacks against other machine learning systems and can serve as an useful tool in measuring the robustness of these systems.

[7] CIFAR10 is a balanced dataset with each class occupying 10% of the test set. We only consider images that are not already in the target class.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *ACM CCS*, 2016.

[2] R. Benenson. What is the class of this image ? `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`, 2016.

[3] E. Biham and A. Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

[4] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Caffe model description: VGG_CNN_S. `https://gist.github.com/ksimonyan/fd8800eeb36e276cd6f9`, 2014.

[5] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014.

[6] A. Fawzi, O. Fawzi, and P. Frossard. Analysis of classifiers' robustness to adversarial perturbations. *CoRR*, abs/1502.02590, 2015.

[7] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard. Robustness of classifiers: from adversarial to random noise. *arXiv preprint arXiv:1608.08967*, 2016.

[8] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Book in preparation for MIT Press, 2016.

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[10] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*, 2016.

[11] S. Gu and L. Rigazio. Towards deep neural network architectures robust to adversarial examples. In *ICLR Workshop*, 2015.

[12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[13] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[14] J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 1997.

[15] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014.

[16] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

[17] P. McDaniel, N. Papernot, and Z. B. Celik. Machine learning in adversarial settings. *IEEE Security & Privacy*, 14(3):68–72, 2016.

[18] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, 2016.

[19] N. Papernot, P. McDaniel, and I. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[20] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Berkay Celik, and A. Swami. Practical black-box attacks against deep learning systems using adversarial examples. In *ACM ASIA CCS*, 2017.

[21] 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. IEEE, 2016.

[22] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy*, 2016.

[23] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015.

[24] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR Workshop*, 2014.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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

[27] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In *Usenix Security*, 2016.

[28] S. Zagoruyko. Cifar-10 in torch. `http://torch.ch/blog/2015/07/30/cifar.html`, 2015.

[29] S. Zagoruyko. Are deep learning algorithms easily hackable? `http://coxlab.github.io/ostrichinator`, 2016.

[30] S. Zagoruyko. Loadcaffe. `https://github.com/szagoruyko/loadcaffe`, 2016.