Pixle: a fast and effective black-box attack based on rearranging pixels

Jary Ponponi[1,*], Simone Scardapane[1], and Aurelio Uncini[1]

[1]Sapienza University of Rome, Via Eudossiana 18, 00184, Rome, Italy
[*]Corresponding author email: jary.pomponi@uniroma1.it

February 7, 2022

Abstract

Recent research has found that neural networks are vulnerable to several types of adversarial attacks, where the input samples are modified in such a way that the model produces a wrong prediction that misclassifies the adversarial sample. In this paper we focus on black-box adversarial attacks, that can be performed without knowing the inner structure of the attacked model, nor the training procedure, and we propose a novel attack that is capable of correctly attacking a high percentage of samples by rearranging a small number of pixels within the attacked image. We demonstrate that our attack works on a large number of datasets and models, that it requires a small number of iterations, and that the distance between the original sample and the adversarial one is negligible to the human eye.

1 Introduction

Neural Networks (NN) have achieved state-of-the-art performance in image classification and many other tasks. However, despite their extensive usage, NNs are highly susceptible to deception performed using adversarial images, which are samples containing small noise, used to fool the NN and force a misclassification. Models that operate in a real world scenario can also be attacked, by physically modifying the objects that have to be classified [1, 2]. This problem is not circumscribed to NNs that operate in the image domain, but also to ones that work over speech, text, or more general domains.

The majority of the existing methods to generate an adversarial image do it by constraining the difference between the original image and the adversarial one, in such a way that the difference between the two is small, and thus hardly visible to the naked eye. The difference between these images is usually calculated using a norm L_p. The value of p influences the

Figure 1: Successful adversarial image from ImageNet using the proposed Pixle attack. The original class of the image is 14 (Passerina Cyanea), while the misclassified class is 883 (Vase). The L_0 distance between the original image and the adversarial one is just 1 pixel in this case.

method used by the algorithm, and we can have: L_0) the difference is pixel wise, by counting the number of pixels that have been modified by the method, L_1) which measures the absolute distance between the images, L_2) that measures the distance as Euclidean distance and, L_∞) which is the largest perturbed pixel in the image.

Another important aspect when developing an attack is the number of iterations required to correctly create an adversarial image. An iteration is an interrogation of the model: the input is passed though it and the output is collected. It is a crucial aspect, because many real life models have a limit on the number of queries that can be performed, and thus it is important to keep this aspect contained.

Moreover, adversarial methods can be grouped into two categories: white-box attacks and black-box attacks [3, 4]. In the first case, which is the most studied one, the methods have full knowledge about the model and its training procedure. In this case, we can rely on any information within the model itself, such as gradients for a given sample or the weights of the model itself. The methods in this category clarify the risks to which the NNs may be exposed, but usually they do not reflect a real case scenario, in which a malicious user wants to attack a model, but has a limited number of queries and no access to the model itself, with the exception of its output. On the other hand, in the black-box case we can access only the input and the output of the model, and thus it is closer to a true case scenario.

In this paper we propose a simple yet effective L_0 norm black-box attack. Our proposal, called Pixle, is capable of efficiently attacking a model by measuring only, given an input sample, the confidence of the prediction, as a probability. Pixle is based on random search, and it creates an effective adversarial image by rearranging a small set of pixels in the image,

showing also that an image usually contains all the pixels that are necessary to misclassify it, using a contained number of iterations, as shown in our experimental results.

2   Related Works

The problem of security in machine learning has always been crucial in order to develop more robust models [5, 6]. The first machine learning model to be attacked were Support Vector Machines [7], then, in [8] and [9], the authors discovered that NNs are prone to such attacks as well, which can be easily accomplished using several gradient based algorithms for obtaining gradient information of the attacked image, and thus fool the model. After these discoveries, the security of NNs has become a critical topic for real world deployments.

Many methods to fool a NN have been proposed in recent years. As said, white-box attacks are the most studied. As attacks based on L_0 norm we have SparseFool [10], which exploits the curvature of decision boundaries, or JSMA [11], that finds the pixels to attack through saliency maps. Based on others norms, we have the Fast Gradient Sign Method [9] and its evolution, called Projected Gradient Descent [12], and Jitter [13], for L_∞ norm; both methods attack the image by changing all the pixels based on their importance, calculated using the gradient associated to the image. For L_1 norm we have attacks that place an upper bound over the absolute sum of the perturbed values, and the most notable examples are DeepFool [14], EAD [15], and FAB [16].

Black-box attacks are usually based on L_2 or L_∞ norms. In [17] the authors proposed a simple policy that produces adversarial images by perturbing random segments of an image to produce the adversarial counterpart. In [17], the authors proposed the idea to identify low-frequency perturbations to improve query efficiency when attacking a model. Another group of methods is composed by approaches that estimate the gradient of the model without accessing it directly [18, 19, 20], emulating white-box attacks. Less studied black-box attacks are the ones based on the L_0 norm. The first method developed in this field is called One-Pixel attack [21], that uses a Differential Evolution (DE) search algorithm [22] to find the best pixels to replace and the values that must overwrite them. It works well on small images, but fails when it is not the case; also, it requires thousands of iterations to find a suitable set of pixels to replace. A similar approach, but based on larger patches, is called PatchAttack [23], that uses reinforcement learning to optimally place pre-computed patches over the image to attack. The drawbacks are that the patches are easily detected, being, often, very large and visible. Another approach, called ScratchThat [24] is also based on DE, and literally 'scratches' the images by adding lines of different colors over them to create perturbed images.

As said, studying how to fool NNs is important to understand also how to protect them from such attacks, and many approaches have been proposed over the years [25]. One way to achieve this goal is to add adversarial images to the training data, such that the robustness against adversarial images can be improved against specific attacks [26, 12]. Others training approaches build more robust models by distillation [27], ensemble of multiple models or models that incorporate some degree of uncertainty within them [28]. In addition, some image processing methods are also proved to be effective in detecting adversarial images.
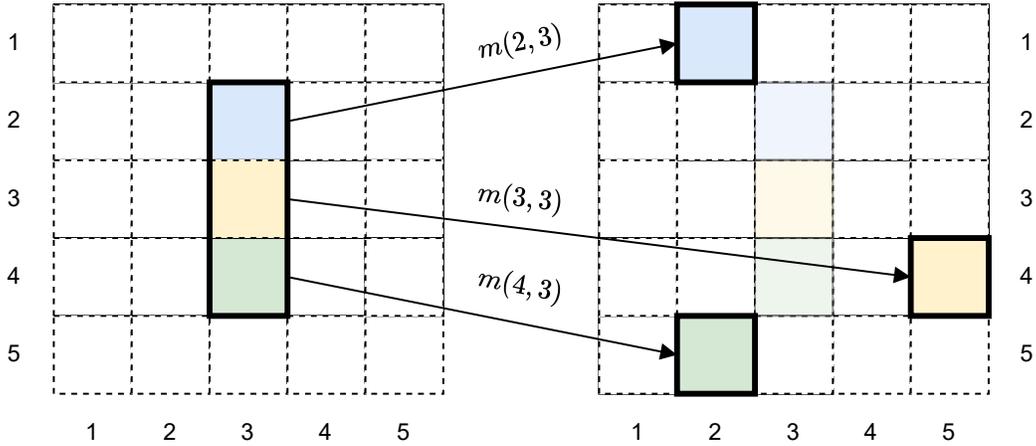
Figure 2: Visualization of the mapping procedure used by Pixle. As the image shows, pixels originating from a small patch of the source image (left) are mapped into the destination image (right), to build the adversarial sample. The pixels in the patch are the only ones that are mapped into new positions, while the others are unchanged. Better viewed in color.

For instance, in [29] the authors studied how the detection of adversarial images can be carried out using noise reduction methods, by comparing the classification after and before applying those techniques. Similarly, in [29] the authors showed that squeezing colors and applying spatial smoothing have high success rate on detecting adversarial images.

3 Methodology

3.1 Problem Description

Generating adversarial images that fool a neural network can be formalized as an optimization problem with constraints.

Let $f : x \in [0,1] \in \mathbb{R}^{(3,w,h)} \to \mathbb{R}^c$ be a classification function that takes as input a sample of size $(3, w, h)$, were 3 are the number of channels while w and h are, respectively, the width and the height of the image, and produces a vector containing c probabilities (such that $\sum_i c_i = 1$), one associated to each class; a sample x can be classified as $\arg\max_i f_i(x)$. In our case, the function f is a NN.

Given an image x, and the associated class y, that is classified correctly by the function f, the goal of an attack is to change the prediction of the function, by producing an adversarial image $\overline{x} \in \mathbb{R}^{(3,w,h)}$ such that $f(\overline{x}) \neq y$. The image $\overline{x}$ must be similar to x, in such a way that the L_p norm of the difference is bounded by a constant ϵ:

$$\arg\max_i f_i(\overline{x}) \neq y \quad ||x - \overline{x}||_p \leq \epsilon \tag{1}$$

with $\epsilon \in \mathbb{N}_+$ in our black-box case. An attack can be untargeted, as described above, or

targeted, by forcing the miss-classification of the adversarial image to a specific adversarial class $\overline{y} \neq y$: $f(\overline{x}) = \overline{y}$. The task of finding the perturbed image $\overline{x}$, associated to x, can be viewed as a minimization problem

$$\min_{\overline{x}} L(f(\overline{x}), y) \quad ||x - \overline{x}||_p \leq \epsilon, \tag{2}$$

for a proper loss function L. Here, we use $L(f(\overline{x}), y) = f_y(\overline{x})$ for untargeted attacks and $L(f(\overline{x}), \overline{y}) = 1 - f_{\overline{y}}(\overline{x})$ for targeted ones; in the first case we want to minimize the score associated to the correct class, while in the second one we want to maximize the score associated to the adversarial target $\overline{y}$.

3.2 Pixle

Pixle is a black-box attack based on random search. Despite its simplicity, random search performs well in many situations and does not depend on gradient information associated to the objective function $L(\cdot, \cdot)$.

Given an image $x \in \mathbb{R}^{(3,w,h)}$, the main idea of the algorithm is to sample a random portion p of adjacent pixels from it, that we call *patch*, and to rearrange some pixels inside it into other positions, calculated, for each pixel, using a predefined function m, that we call the *map* function. Given an image x, a generic patch is defined by a 4-tuple $p = (w_p, o_y, w_p, h_p)$, where $0 < o_x \leq w$ and $0 < o_y \leq h$ are the coordinates on the image used as origin point of the patch, while w_p and h_p are, respectively, the weight and the height of the patch. We indicate as P the list of index tuples (i, j), that are defined by the coordinates of the patch, and thus are contained in the rectangle having vertices as $[(o_x, o_y), (o_x + w_p, o_y), (o_x, o_y + h_p), (o_x + w_p, o_y + h_p)]$. We formalize this list as $P = \left[(o_x + i, o_y + j) \right]_{\forall i \in \{0, ..., w_p\}, j \in \{0, ..., h_p\}}$, which has cardinality $|P| = w_p * h_p$, assuming that the patch does not exceeds the boundary of the image, otherwise it is moved in such a way that all the indexes are inside the image. The mapping function m is defined as:

$$m \colon (i, j) \in P \mapsto \mathbb{N}_+^2 \in [1, h] \times [1, w], \tag{3}$$

that returns, for each pixel in the patch, the position where the pixel must be moved in the original image. Given this function, we can define each pixel of the adversarial image as:

$$\overline{x}_{i,j} = \begin{cases} m(i, j) & (i, j) \in P \\ x_{i,j} & \text{otherwise} \end{cases}$$

The exposed procedure changes only the destination pixels, that are overwritten by the source ones. This approach is useful if we want to search for an adversarial image that reduces the distance from the original one. If we don't care about this aspect, and we want to speed up the convergence, another possible approach is to swap the position of the source and destination pixels at the same time, using the mapping function as before. In this way, no redundant information is injected into the adversarial image. The overall procedure,

---

Algorithm 1 Pixle: Restart-Iterative algorithm

Require: input image x with its associated label y. Maximum and minimum dimension for source patch. The number of restarts R and the iterations to perform for each restart step T. The mapping function m.

Ensure: $y = x^n$

    $\overline{x} \leftarrow x$
    $l \leftarrow f_y(x)$
    for $r = 0$ to R **do**
        $x^r \leftarrow \overline{x}$
        for $t = 0$ to T **do**
            Sample $p = (o_x, o_y, w_p, h_p)$.
            Calculate the set P
            $x^t \leftarrow \overline{x}$
            for $\forall (i,j) \in P$ **do**
                $(z, k) \leftarrow m(i, j)$
                $x^t_{z,k} \leftarrow x_{i,j}$
            end for
            if $f_y(x^t) < l$ **then**
                $l \leftarrow f_y(x^t)$
                $x^r \leftarrow x^t$
            end if
        end for
        $\overline{x} \leftarrow x^r$
    end for
        return $\overline{x}$

---

sampling a patch and calculating the mapping function codomain, remains the same. Fig. 2 visually shows how the pixels are mapped from the source image to the destination one.

3.3 Implementing the mapping function

The function m can be implemented in different ways. Here, since we focus on the speed of the attack and the number of times that the models is interrogated, we propose and test the following implementations:

- Random: the function m returns, for each input pixel, a random coordinate point ($i \sim \mathbb{U}[1, h], j \sim \mathbb{U}[1, w]$), different from the origin position.

- Similarity: given the image x and the pixel i extracted using an index tuple from P, it returns the position of the most similar pixel in the image that is not equal to i.

- Distance: it works like the Similarity approach, but in this case the position of the most different pixel is selected, avoiding the ones that have zero distance.

Algorithm 2 Pixle: Iterative algorithm

Require: input image x with its associated label y. Maximum and minimum dimension for source patch. The number of iterations to perform for each restart step T. The mapping function m.
Ensure: $y = x^n$
   $\overline{x} \leftarrow x$
   $l \leftarrow f_y(x)$
   $x \leftarrow \overline{x}$
   for $t = 0$ to T **do**
      Sample $p = (o_x, o_y, w_p, h_p)$.
      Calculate the set P
      $x^t \leftarrow \overline{x}$
      for $\forall (i,j) \in P$ **do**
         $(z, k) \leftarrow m(i, j)$
         $x^t_{z,k} \leftarrow x_{i,j}$
      end for
      if $f_y(x^t) < l$ **then**
         $l \leftarrow f_y(x^t)$
         $\overline{x} \leftarrow x^i$
      end if
   end for
      return $\overline{x}$

- Similarity Distribution: like the similarity approach, this is based on the most similar pixels, but, instead of selecting the position in a deterministic way, a distribution of positions is calculated using all the distances between the source pixel and all the others; in this way, the most similar pixels have a higher probability of being selected. Once the distribution has been calculated, the mapped position is sampled from it.

- Distance Distribution: it is the distance based version of the Similarity Distribution exposed above, where most distant pixels are more likely to be selected.

3.4 Search Algorithms

We propose two different procedures to find the adversarial image. In the first one, we have two loops, one nested inside the other, and we select only the attack that leads to the highest decrease of the loss; this attack is called Restart-Iterative. In the second one, each pixel replacement is used to create the final adversarial image, and it is called Iterative.

    Restart-Iterative: the algorithm is composed of a fixed number of restarts $R \geq 1$, and within each one a maximum number of iterations M are performed. Before starting the main loop, the adversarial image is a copy of the original one, so that all the pixels from the original image are preserved during the process. At every iteration, we sample a source patch

p, and then, using the mapping function m, the pixels in the set P are overwritten. Only the mapping that decreases the loss the most is actually saved and used in the next restart step. This algorithm is summarized in Alg. 1.

Iterative: in this version, we have only the internal iteration loop, and the adversarial image is updated each time that an attack decreases the loss value. The algorithm is summarized in Alg. 2.

Intuitively, the first approach requires more iterations, because only the best attacks are saved and preserved, while the second approach updates the adversarial image after each attack. The latter could require less iterations, but it can happen that an attack leads to a region of the search space which is sub-optimal, due to the lack of any controlling strategy.

4   Experimental evaluation

4.1   Experimental setup

The evaluation of the proposed attack is carried out on the following datasets: CIFAR10 [30], TinyImagenet, and ImageNet [31]. We train CIFAR10 and TinyImagenet using, respectively, ResNet20 [32] and VGG11 [33] for CIFAR10 and ResNet50 [32] and VGG16 [33] for TinyImageNet. Regarding ImageNet, we used the pretrained version of ResNet50 [32] and VGG16 [33], without fine-tuning them.

As training procedure for CIFAR10 and TinyImageNet, we use Stochastic Gradient Descent, with learning rate set to 0.01 and momentum to 0.9. Also, we use the standard augmentation schema: the images are flipped horizontally with a probability of 0.5, and random cropped to the same original size, but after applying a padding.

For each experiment, we extract the test images to attack from the subset of test images that are correctly classified by the models. For CIFAR10 we use 100 images per each class, while for TinyImageNet and ImageNet, we use, respectively, 5 and 1 images per class. In this way, we test the attacks on the same number of images for each dataset (1000). We also test the ability of the approaches to perform targeted attacks. For this purpose, we use 20 images per class for CIFAR10, resulting in 200 attacks, excluding the other datasets.

We compare our proposal to two others black-box attacks based on the L_0 norm: ScratchThat [24] and OnePixel attack [21]. For the latter, we based our implementation on the one present in TorchAttacks [34], while the first one is a custom implementation [1]. For each attack we performed a search of the best hyper-parameters, based on the results from the respective papers, in order to produce the best results in the smallest number of iterations. For ScratchThat, we use 1 Bézier Curve for CIFAR10 and 3 otherwise, with a population size of 50 and maximum number of iterations set to 50 for the DE algorithm. While for OnePixel, we set the number of pixels to modify equals to 5, and, regarding the DE algorithm, we set the population size to 100 solutions and the number of iterations to 50. Regarding our approach, we use the Restart-Iterative approach as random search with at most 100 restarts and 50 iterations per restart cycle. The patches have a size of 3 pixels, and the mapping

---

[1]The complete code used to run all the experiments can be found here

function is the random one, which is calculated again at each iteration. All the methods are interrupted when the current image is misclassified by the model, for non targeted attacks, or when the image is classified as the target class, for targeted ones.

In order to compare the approaches, we use the following metrics:

- Success Rate: it is defined as the percentage of adversarial images that are misclassified by the neural network.

- Iterations: calculated image wise, it is the number of times that a model is interrogated while searching the adversarial counterpart of that image.

- L_0 norm: the number of pixels that the approach has modified.

4.2 Main results

4.2.1 Non targeted attacks

The results in Table 1 shows that our approach is the most performant one across the vast majority of scenarios. In fact, it is capable of achieving almost 100% of success rate on all the datasets, while the others approaches fail to achieve comparable results when the images become bigger.

Regarding the iterations, we see that our proposal requires a lower number of iterations even when the dimensions of the images increase with the exception of OnePixel, which requires fewer iterations than the others, but the success rate is not sufficiently high to be considered a good attack for big images. Figure 3 shows how the loss value decreases when attacking CIFAR10 and ImageNet using our proposal. The images contain the loss value for each attacked image, as well as the average loss on each iteration (red dots). According to the images, it can be seen that a set of images are easily attacked and the associated loss values rapidly decrease, as the bottom images also show; this happens for both CIFAR10 and ImageNet. In fact, for CIFAR10 our approach correctly attacks 80% of the images after 600 iterations, while the same number of images from ImageNet are correctly attacked after 750 iterations (approximately).

In the end we analyze the L_0 norm. The same table shows that our approach changes less pixels than ScratchThat, when attacking all the dataset. Also, if we cross reference the results in terms of L_0 norms and Fig. 3, we see that a lot of images are correctly attacked using a small number of pixels, while other images require more iterations (as also shown by the standard deviation of the metrics). Moreover, the hyperparamenters of our approach can be adapted in order to achieve a smaller L_0 loss, at the expense of the number of iterations or success rate (as shown in Table 2).

4.2.2 Targeted attack

Figure 4 shows the results obtained when attacking each class of CIFAR10 with respect to all the others classes. It shows that OnePixel is incapable of attacking most of the classes, and

Table 1: Results obtained when attacking multiple datasets trained on ResNet and VGG architectures. For each score, we show the mean and the variance, if present, calculated over all the images on that dataset. Best results for each pair dataset-architecture are highlighted in bold.

Dataset	Model	Method	Success rate	Iterations	L_0 norm
CIFAR10	ResNet18	OnePixel	64.7	$5125_{\pm799}$	5
		ScratchThat	99.7	1010	$38.3_{\pm5.2}$
		Pixle (Proposed)	**100**	$119_{\pm141}$	$26.8_{\pm22.8}$
	VGG11	OnePixel	84.5	5100	5
		ScratchThat	99.3	1010	$27.64_{\pm4.8}$
		Pixle (Proposed)	**100**	$80_{\pm145}$	$20.1_{\pm21.9}$
TinyImageNet	ResNet50	OnePixel	21.0	5100	5
		ScratchThat	69.7	1010	$49.1_{\pm2.2}$
		Pixle (Proposed)	**99.6**	$310_{\pm561}$	$59.0_{\pm88.5}$
	VGG16	OnePixel	31.9	5100	5
		ScratchThat	76.6	1010	$48.6_{\pm2.4}$
		Pixle (Proposed)	**100**	$87_{\pm201}$	$21.5_{\pm30.6}$
ImageNet	ResNet50	OnePixel	47.7	5100	5
		ScratchThat	82.6	$623_{\pm321}$	$175.2_{\pm25.7}$
		Pixle (Proposed)	**98.0**	$341_{\pm426}$	$155.7_{\pm184.2}$
	VGG16	OnePixel	31.9	5100	5
		ScratchThat	81.8	$753_{\pm156}$	$143.0_{\pm6.0}$
		Pixle (Proposed)	**99.0**	$519_{\pm780}$	$98.5_{\pm137.5}$

Table 2: The results obtained when varying the side dimension of a patch, for each combination of dataset and architecture, using Pixle.

Dataset	Model	Patch dimension	Success rate	Iterations	L_0 norm
CIFAR10	ResNet18	1	100	$314_{\pm311}$	$6.9_{\pm6.1}$
		3	100	$119_{\pm141}$	$26.8_{\pm22.8}$
		5	100	$78_{\pm123}$	$51.6_{\pm44.8}$
	VGG11	1	99.1	$343_{\pm726}$	$7.3_{\pm13.6}$
		3	100	$80_{\pm145}$	$20.1_{\pm21.9}$
		5	100	$56_{\pm236}$	$38.2_{\pm33.9}$
TinyImageNet	ResNet50	1	96.81	$754_{\pm1055}$	$15.3_{\pm20.0}$
		3	99.6	$310_{\pm561}$	$59.0_{\pm88.5}$
		5	99.8	$182_{\pm379}$	$98.0_{\pm137.7}$
	VGG16	1	99.6	$242_{\pm172}$	$18.0_{\pm29.9}$
		3	100	$87_{\pm201}$	$21.5_{\pm30.6}$
		5	100	$52_{\pm120}$	$43.2_{\pm49.6}$
ImageNet	ResNet50	1	94.5	$1417_{\pm1376}$	$28.9_{\pm27.3}$
		3	98.0	$341_{\pm426}$	$155.7_{\pm184.2}$
		5	99.5	$223.3_{\pm315}$	$286.2_{\pm372.3}$
	VGG16	1	95.6	$1194_{\pm1319}$	$24.4_{\pm26.1}$
		3	99.0	$519_{\pm780}$	$98.5_{\pm137.5}$
		5	99.5	$339_{\pm609}$	$183.9_{\pm291.7}$

presents a low success rate. Regarding Scratch That, it is capable of perfectly attacking most of the pairs source-destination, achieving half of the times a perfect score. Our approach, instead, achieves a perfect score on almost all the attacks, with the exception of some couples.
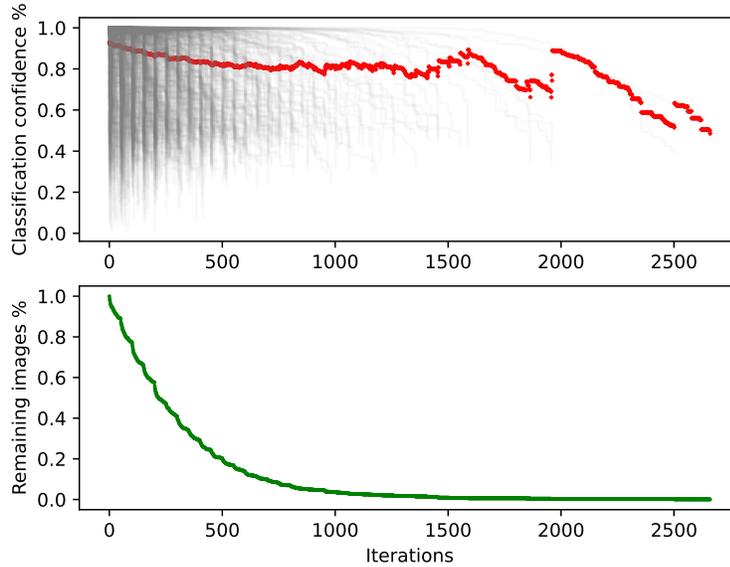
4.2.3 Dimension of the patches

A crucial aspect of our proposal is the dimension of the patches, and in Table 2 we study this aspect, by calculating all the metrics while varying the dimensions of them. For each experiment, we use random mapping and at most 100 restart, each one composed of 50 iterations.
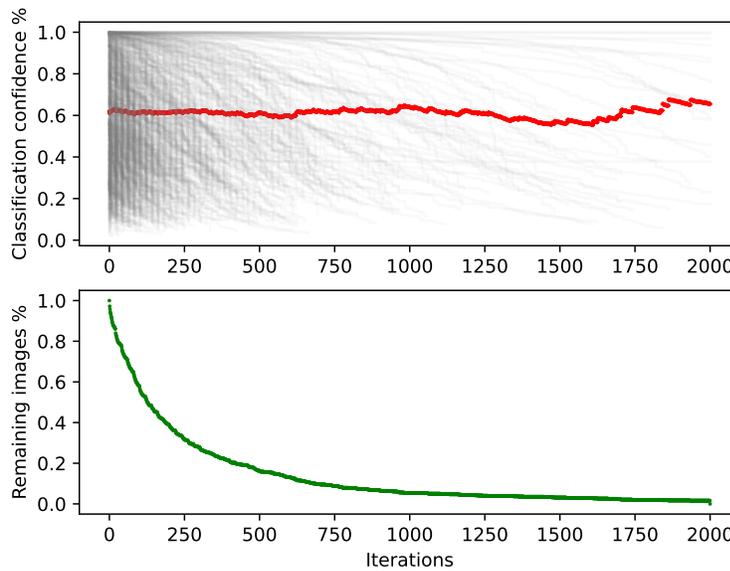
We see that, as expected, the bigger the patches are the fewer iterations are required, but at the same time the distance between the original image and the adversarial counterpart grows. This happens because more pixels are moved at the same time, leading rapidly to a minima, which is accepted due to the callback which interrupts the searching process, that could be reached by moving fewer pixels as well (this is clearly visible by comparing the results obtained using 3 pixels with the one achieved when using 5 pixels).

4.3 Ablation studies

We perform an ablation study to show how the individual design decisions improve the performance of Pixle, both in terms of scores and convergence.

(a) CIFAR10 results using VGG11.

(b) ImageNet results using ResNet50.

Figure 3: Each figure shows, on top, how the losses (the probability associated to the correct class) change during the iterations of our proposal, using the Restart-Iterative algorithm (the red dots are the average loss calculated on that iteration); while the bottom image shows how many images are left to attack after each iteration. Better viewed in colors.

Table 3: The results obtained using VGG11 trained on CIFAR10. For the Iterative-Restart approach we fixed the number of iterations to 59.

	Restart-Iterative (restarts)				Iterative (iterations)						
	50	100	250	500	50	100	250	500	1000	2000	10000
Success Rate	95.5	98.2	99.7	100.0	6.1	77.7	87.9	90.5	92.4	93.1	94.3
Iterations	$170_{\pm243}$	$199_{\pm362}$	$222_{\pm531}$	$209_{\pm468}$	$31_{\pm17}$	$45_{\pm36}$	$69_{\pm79}$	$95_{\pm147}$	$131_{\pm264}$	$206_{\pm511}$	$697_{\pm2365}$
L_0 norm	$8.9_{\pm11.6}$	$10.2_{\pm16.9}$	$10.9_{\pm22.5}$	$10.9_{\pm20.3}$	$12.2_{\pm9.1}$	$16.1_{\pm13.7}$	$18.7_{\pm18.9}$	$20.4_{\pm23.0}$	$20.4_{\pm23.4}$	$21.1_{\pm25.5}$	$21.86_{\pm28.2}$

Table 4: The results obtained when varying the mapping function. The attacked model is VGG11 trained on CIFAR10.

Mapping function	Success rate	Iterations	L_0 norm
Distance	17.6	$8297_{\pm3697}$	$40.1_{\pm17.9}$
Similarity	96.5	$1029_{\pm2196}$	$10.0_{\pm18.7}$
Distance distribution	99.3	$536_{\pm1260}$	$5.9_{\pm11.9}$
Similarity distribution	99.1	$605_{\pm1425}$	$3.1_{\pm7.3}$

4.3.1 Restart-Iterative vs Iterative

In this section we explore how the choice of the search algorithm affects the results. Table 4.3 shows the results obtained training VGG11 on CIFAR10, while changing the number of iterations. To this end, the algorithm attacks 1 pixel at a time, using a random mapping calculated at each iteration. We can see that the random search based on the restarts is more effective with respect to all the metrics. In fact, when setting the limit to 100 iterations and 50 restarts, we achieve better results then the iterative algorithm with a limit of 10000 iterations.

4.3.2 Mapping function

In this section we study how the selection of mapping function affects the final scores, shown in Table 4. We see that, when operating with a deterministic mapping, the number of required iterations, as well as the L_0 norm, are higher than the distribution counterparts. This happens because the randomness of the distributions force the approach to explore the space, avoiding local minima.

5 Conclusion

In this paper we proposed a novel black-box attack called Pixle. Our attack, despite using only the probability of the prediction returned by the attacked model, proved to be effective over a wide range of datasets and architectures types, and on multiple metrics, such as the number of iterations required and the number of modified pixels.

As future work, we aim to understand the correlation between the attacked class and the pixels in the image, especially when performing a targeted attack. Furthermore, we would like to expand the proposed approach also on other domains, such as text and audio. In the
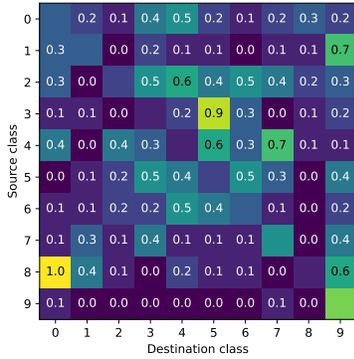
end, we would like to understand if the proposed attack can be further studied, in order to create a defense algorithm against L_0 attacks.
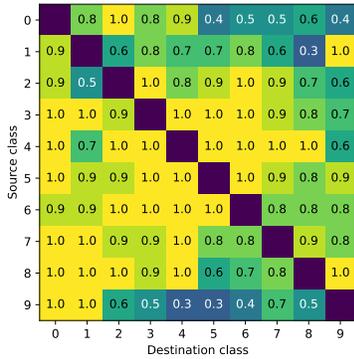
References

[1] A. Athalye, L. Engstrom, A. Ilyas, K. Kwok, Synthesizing robust adversarial examples, in: International conference on machine learning, PMLR, 2018, pp. 284–293.

[2] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, D. Song, Robust physical-world attacks on deep learning visual classification, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 1625–1634.

[3] S. Bhambri, S. Muku, A. Tulasi, A. B. Buduru, A survey of black-box adversarial attacks on computer vision models, arXiv preprint arXiv:1912.01667 (2019).

[4] S. Qiu, Q. Liu, S. Zhou, C. Wu, Review of artificial intelligence adversarial attack and defense technologies, Applied Sciences 9 (5) (2019).

[5] M. Barreno, B. Nelson, A. D. Joseph, J. D. Tygar, The security of machine learning, Machine Learning 81 (2) (2010) 121–148.

[6] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, J. D. Tygar, Can machine learning be secure?, in: Proceedings of the 2006 ACM Symposium on Information, computer and communications security, 2006, pp. 16–25.

[7] B. Biggio, B. Nelson, P. Laskov, Poisoning attacks against support vector machines, arXiv preprint arXiv:1206.6389 (2012).

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: 2nd International Conference on Learning Representations, ICLR 2014, 2014.

[9] I. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in: International Conference on Learning Representations, 2015, pp. 1–11.

[10] A. Modas, S.-M. Moosavi-Dezfooli, P. Frossard, Sparsefool: a few pixels make a big difference, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 9087–9096.

[11] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, A. Swami, The limitations of deep learning in adversarial settings, in: 2016 IEEE European symposium on security and privacy (EuroS&P), IEEE, 2016, pp. 372–387.

[12] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, in: International Conference on Learning Representations, 2018.

[13] L. Schwinn, R. Raab, A. Nguyen, D. Zanca, B. Eskofier, Exploring robust misclassifications of neural networks to enhance adversarial attacks, arXiv e-prints (2021) arXiv–2105.

[14] S.-M. Moosavi-Dezfooli, A. Fawzi, P. Frossard, Deepfool: a simple and accurate method to fool deep neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2574–2582.

[15] Y. Sharma, P.-Y. Chen, Attacking the madry defense model with l_1-based adversarial examples, arXiv preprint arXiv:1710.10733 (2017).

[16] F. Croce, M. Hein, Minimally distorted adversarial examples with a fast adaptive boundary attack, in: International Conference on Machine Learning, PMLR, 2020, pp. 2196–2205.

[17] N. Narodytska, S. Kasiviswanathan, Simple black-box adversarial attacks on deep neural networks, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2017, pp. 1310–1318.

[18] A. Ilyas, L. Engstrom, A. Athalye, J. Lin, Black-box adversarial attacks with limited queries and information, in: International Conference on Machine Learning, PMLR, 2018, pp. 2137–2146.

[19] M. Alzantot, Y. Sharma, S. Chakraborty, H. Zhang, C.-J. Hsieh, M. B. Srivastava, Genattack: Practical black-box attacks with gradient-free optimization, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2019, pp. 1111–1119.

[20] A. N. Bhagoji, W. He, B. Li, D. Song, Exploring the space of black-box attacks on deep neural networks, arXiv preprint arXiv:1712.09491 (2017).

[21] J. Su, D. V. Vargas, K. Sakurai, One pixel attack for fooling deep neural networks, IEEE Transactions on Evolutionary Computation 23 (5) (2019) 828–841.

[22] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, Journal of global optimization 11 (4) (1997) 341–359.

[23] C. Yang, A. Kortylewski, C. Xie, Y. Cao, A. Yuille, Patchattack: A black-box texture-based attack with reinforcement learning, in: European Conference on Computer Vision, Springer, 2020, pp. 681–698.
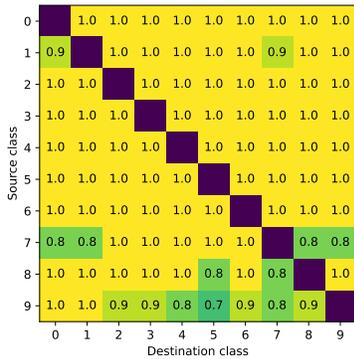
[24] M. Jere, L. Rossi, B. Hitaj, G. Ciocarlie, G. Boracchi, F. Koushanfar, Scratch that! an evolution-based adversarial attack against neural networks, arXiv preprint arXiv:1912.02316 (2019).

[25] X. Yuan, P. He, Q. Zhu, X. Li, Adversarial examples: Attacks and defenses for deep learning, IEEE transactions on neural networks and learning systems 30 (9) (2019) 2805–2824.

[26] R. Huang, B. Xu, D. Schuurmans, C. Szepesvári, Learning with a strong adversary, arXiv preprint arXiv:1511.03034 (2015).

[27] N. Papernot, P. McDaniel, X. Wu, S. Jha, A. Swami, Distillation as a defense to adversarial perturbations against deep neural networks, in: 2016 IEEE symposium on security and privacy (SP), IEEE, 2016, pp. 582–597.

[28] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, et al., A survey of uncertainty in deep neural networks, arXiv preprint arXiv:2107.03342 (2021).

[29] B. Liang, H. Li, M. Su, X. Li, W. Shi, X. Wang, Detecting adversarial image examples in deep neural networks with adaptive noise reduction, IEEE Transactions on Dependable and Secure Computing (2018).

[30] A. Krizhevsky, G. Hinton, et al., Learning Multiple Layers of Features from Tiny Images (2009).

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255.

[32] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[33] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv preprint arXiv:1409.1556 (2014).

[34] H. Kim, Torchattacks: A pytorch repository for adversarial attacks, arXiv preprint arXiv:2010.01950 (2020).

(a) The results obtained using OnePixel attack.

(b) The results obtained using Scratch That attack

(c) The results obtained using our proposal.

Figure 4: The images show the success rate on ResNet18 trained using CIFAR10. The results are calculated using 20 test images, classified correctly by the model, for each class. Each matrix contains the percentage of attacks that have been successfully completed.