Adversarial scratches: Deployable attacks to CNN classifiers

Loris Giulivi [a],[*], Malhar Jere [b], Loris Rossi [a], Farinaz Koushanfar [b], Gabriela Ciocarlie [c], Briland Hitaj [d], Giacomo Boracchi [a]

[a] Politecnico di Milano, Piazza Leonardo da Vinci, 32, Milano 20133, Italy
[b] University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0021, USA
[c] University of Texas San Antonio, One UTSA Circle, San Antonio, TX 78249, USA
[d] SRI International, 333 Ravenswood Ave Menlo Park, CA 94025, USA

ABSTRACT

A growing body of work has shown that deep neural networks are susceptible to adversarial examples. These take the form of small perturbations applied to the model's input which lead to incorrect predictions. Unfortunately, most literature focuses on visually imperceivable perturbations to be applied to *digital images* that often are, by design, impossible to be deployed to physical targets.

We present Adversarial Scratches: a novel L_0 black-box attack, which takes the form of scratches in images, and which possesses much greater *deployability* than other state-of-the-art attacks. Adversarial Scratches leverage Bézier Curves to reduce the dimension of the search space and possibly constrain the attack to a specific location.

We test Adversarial Scratches in several scenarios, including a publicly available API and images of traffic signs. Results show that our attack achieves higher fooling rate than other deployable state-of-the-art methods, while requiring significantly fewer queries and modifying very few pixels.

© 2022 Elsevier Ltd. All rights reserved.

1. Introduction

Convolutional Neural Networks (CNN) [15] have achieved state-of-the-art performance on a wide array of tasks. These models, however, are surprisingly susceptible to deception by adversarial examples [29], consisting in small perturbations to the input that lead to incorrect predictions. The relevance of this security issue is made clear by the increased number of critical systems that make use of CNNs. A plethora of papers have explored adversarial vulnerabilities in neural networks, such as Croce et al. [6], Narodytska and Kasiviswanathan [22], giving rise to a large corpus of attacks. Typically, attacks are composed of two key components: the perturbation model and the search strategy. The former relates to how the image is modified (e.g. the attack only affects a square patch), the latter regards how a successful perturbation is computed, typically requiring an iterative optimization procedure. Most of the literature, however, focuses on attacks designed for *digital images*. As such, perturbations are confined to the digital domain, and are impossible to be deployed to a physical target.

Recently, there has been a surging research interest on attacks that can be deployed on real-world systems [9]. Following this line, in our work we present attacks whose perturbations are designed to be deployable. We define attacks to be *deployable* based on two required characteristics:

C1: the perturbation only affects the image's pixels that belong to a specific target object;
C2: the perturbation is contained in a small, spatially adjacent region.

When an attack does not meet the first condition, it means that the perturbation spans multiple objects in the image, possibly including the background. In this case, the attack cannot be applied to a physical target (for example, $L1, L2, L_\infty$ and Frame attacks in Fig. 1). When the second condition is not met, the attack may modify a multitude of regions in the target. Clearly, accurately accounting for the relative position of these regions on the target is hardly feasible, rendering the attack non-deployable (for example, the Sparse attack in Fig. 1). Attacks such as Croce et al. [6], Yang et al. [31] are some of the rare examples that satisfy both (C1) and (C2), and rely on localizing the perturbation inside a square patch (Patch attack in Fig. 1). However, these either require many attempts (high query requirement) [6] or large patches [31] to be successful.

In this work, we propose *Adversarial Scratches*: a novel, powerful, deployable attack, illustrated in the Scratch attack in Figs. 1 and 2. Adversarial Scratches are constituted of parametric curves, resembling graffiti or small damages when applied to

---

* Corresponding author.
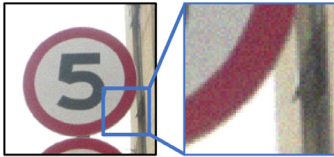 *E-mail address:* loris.giulivi@polimi.it (L. Giulivi).
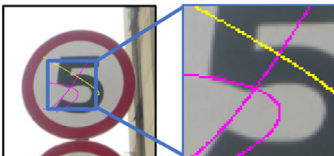
Deployable and non-deployable perturbations

Fig. 1. Example adversarial attacks on the TSRD traffic sign dataset. Most perturbations designed for digital images cannot be applied to a physical target. In our work, we focus on deployable perturbations.

Fig. 2. Successful Adversarial Scratches on ImageNet (one scratch, $L_0 = 50$). Adversarial Scratches are powerful attacks that both require a minimal perturbation (small L_0) and a low number of queries to be successful. On top of this, the adjacency of the perturbed region enables this attack to be deployable in the physical world.

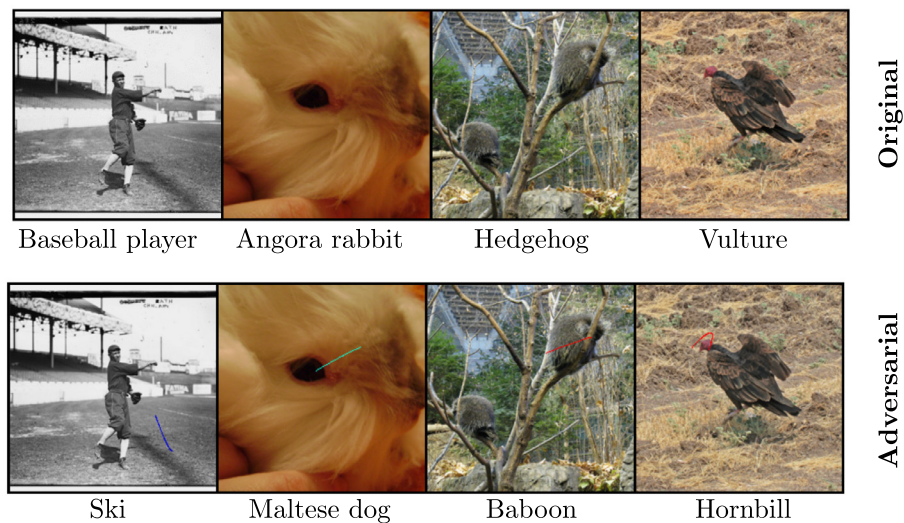a target, and are thus spatially adjacent (C2). In particular, we make use of Bézier curves, as these can express a wide variety of shapes. Our intuition is that these curves may introduce patterns in the image to which CNNs are sensitive. Moreover, their compact parametric representation allows efficient optimization. Crucially, Bézier curves can be arbitrarily clipped (see Section 4.1), allowing to confine the attack to the target region (C1). Lastly, Adversarial Scratches are set in the black-box attack scenario, meaning that the model's internals are not known when computing the attack. As such, Adversarial Scratches leverage a variety of gradient-free search strategies. Our contributions are the following:

- **Adversarial Scratches:** We introduce a new perturbation model which is deployable by design. Adversarial Scratches only perturb small regions in the image in the shape of Bézier curves. Furthermore, the parametric nature of Adversarial Scratches enables to achieve state-of-the-art fooling rates while greatly reducing query requirements.
- **Countermeasures:** We propose two countermeasures to our attacks, namely, median filtering and JPEG compression, and assess their impact in mitigating the effects of Adversarial Scratches.
- **Adversarial software framework:** We release a Python open-source tool to design and perform adversarial attacks on several popular CNNs for image classification. Within our framework, we also provide interfaces for various optimization strategies, perturbation models, and target networks.
- **A benchmark for deployable attacks:** We manually segment target regions for a subset of samples of the TSRD [23] dataset, and make these segmentations publicly available. The improved dataset poses as a benchmark for simulation of deplyable attacks on *in-the-wild* traffic signs.

Our code and datasets are made publicly available at https://github.com/loris2222/AdversarialScratches.

Our extensive experimental evaluation analyzes the performance of Adversarial Scratches across a variety of scenarios. Firstly, we consider a well-established test-bed on ImageNet, and compare against other deployable and non-deployable state-of-the-art attacks. Secondly, we design an experiment utilizing our own-developed version of the TSRD dataset, where perturbations are applied to images of traffic signs. Furthermore, we launch an attack against Microsofts commercially available Cognitive Services Image Captioning API. Our attacks successfully fooled the API, demonstrating the effectiveness of Adversarial Scratches on a production-grade Machine-Learning-as-a-service system. *We have contacted Microsoft regarding this vulnerability*. Lastly, we study the performance of Adversarial Scratches across a variety of optimization strategies and parametric configurations.

2. Background and related work

After providing a formal description of the problem, we categorize adversarial attacks on neural networks and overview the state-of-the-art.

2.1. Problem formulation

We denote as $\mathbf{x} \in [0,1]^{w,h,c}$ an image having width w, height h, and c channels, and as $f(\cdot)$ an image classifier (a CNN in our case). We assume the CNN to return a vector $f(\mathbf{x})$ where each component $f(\mathbf{x})_i$ represents the posterior probability of $\mathbf{x}$ belonging to class i, thus, the label assigned by the CNN to $\mathbf{x}$ is $C(\mathbf{x}) = \text{argmax}_i f(\mathbf{x})_i$. We denote the ground truth label as y.

Given a correctly classified sample $\mathbf{x}$, namely, $C(\mathbf{x}) = y$, the problem consists in finding an adversarial sample $\mathbf{x}' \in [0,1]^{w,h,c}$ which, according to a distance metric L and a distance threshold δ, is close to the original sample $\mathbf{x}$:

$$L(\mathbf{x}', \mathbf{x}) < \delta, \tag{1}$$

such that *the model is fooled*, which means:

$$C(\mathbf{x}') \neq y. \tag{2}$$

Typically, the distance L is realized in the L_p norm of the difference between the original and the adversarial samples, namely $L(\mathbf{x}', \mathbf{x}) = \|\mathbf{x}' - \mathbf{x}\|_p$. We consider the L_0 norm as this is the only metric enabling spatially contained perturbations, and in turn satisfying (C2) (Fig. 1). Another requirement for deployability of the attack is constraining the perturbation to the target's surface (C1). Therefore, for each image $\mathbf{x}$, we also require a target region $\mathbf{r}$ indicating the pixels of $\mathbf{x}$ that belong to the target object. Fig. 3 shows how this region is used.

2.2. Prior works and categorization of attacks

Biggio et al. [2,3] were the first to present adversarial examples in gradient-based learning systems, such as support vector machines (SVMs) and neural networks. Szegedy et al. [29] discovered that this issue also extends to ImageNet-trained deep neural networks. Other works show that adversarial examples can target models addressing tasks other than classification, such as clustering [5]. Attacks to neural networks are characterized by two major ingredients: *i)* the *perturbation model*, which places constraints on the attack, thereby defining the feasible search space for $\mathbf{x}'$, and *ii)* the *search strategy* used to explore this space.
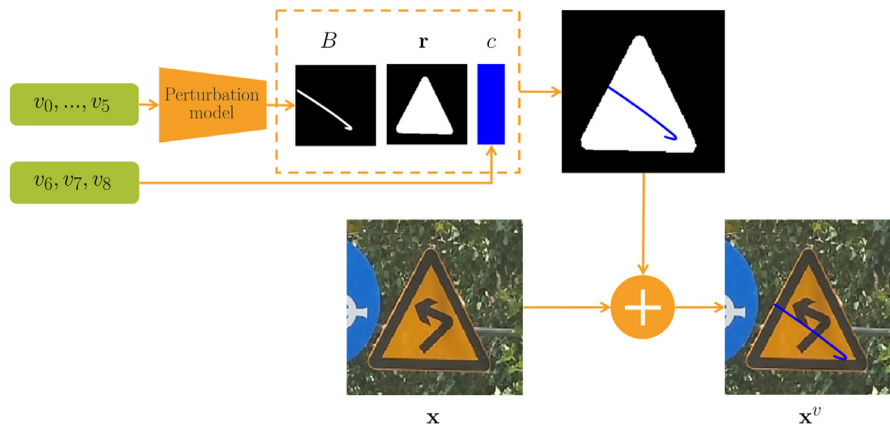

Fig. 3. Attack application methodology. From the parametric description of the attack, a perturbation can be constructed, masked, and then applied to the image, thus obtaining the adversarial image.

Table 1
Comparison of methodologies in the literature and our own.

Method	Search strategy	Perturbation structure	Large network (ImageNet)	Localized Perturbation (L0 limited)?	Deployable attack?
Narodytska and Kasiviswanathan [22]	local random search	sparse	✓	✓	✗
JSMA [24]	substitute model	sparse	✗	✓	✗
Boundary attack [4]	Gaussian perturbation	unstructured	✓	✗	✗
Ilyas et al. [14]	natural evolution strategy	unstructured	✓	✗	✗
CornerSearch [7]	coordinate-wise gradient estimation	sparse	✗	✓	✗
SimBA [11]	Fourier coefficients	unstructured sparse	✓	✓	✗
Parsimonious Black-Box Attacks [21]	greedy local search	unstructured	✓	✗	✗
PatchAttack [31]	RL agent	patch	✓	✓	✓
DEceit [10]	differential evolution	sparse	✓	✓	✗
Sparse-RS [6] ("any-pixel")	random search	sparse	✓	✓	✗
Sparse-RS [6] (Patch-RS)	random search	patch	✓	✓	✓
Adversarial Scratches (Ours)	NGO	scratch	✓	✓	✓

Perturbation models can be categorized along four axes. The first regards the access level the attacker has on the target neural network (i.e., attack surface), the second specifies how the perturbation magnitude is measured (i.e., metric), the third regards specific constraints on the support of the perturbation (i.e., geometric structure), and the fourth denotes the aim of the attack.

Attack surface: along this axis, perturbation models are categorized in white-box and black-box. The first category indicates that the attacker has a full view of the internals of the model f, including its gradients, the latter instead specifies that the attacker may only control the input $\mathbf{x}'$ and observe its output. Black-box attacks can also be decomposed into *score-based* attacks, when the full class score vector $f(\mathbf{x}')$ is provided [11], or *decision-based* ones [4], when only the predicted label $C(\mathbf{x}')$ is provided. An example of a white-box attack is the JSMA attack [24], which finds vulnerable pixels through saliency maps. Regarding black-box attacks, we mention DEceit [10], which uses differential evolution to optimize an attack with adjustable sparsity.

White-box attacks pose no serious threats to production-level systems, as it is unlikely that providers would disclose information regarding their models. Thus, we choose to frame ourselves in the black-box setting, and allow our search strategy to only control the model input and have access only to its output.

Metric: On the second axis, perturbation models are categorized depending on the particular L_p norm chosen to constrain the magnitude of the applied perturbation, which is measured by $\|\mathbf{x}' - \mathbf{x}\|_p$. L_0 attacks measure and regulate the number of perturbed pixels, L_1 and L_2 attacks the Manhattan and Euclidean distance between the original and the perturbed image, and L_∞ attacks the largest pixel-wise difference between the two. An adversarial sample $\mathbf{x}'$ is deemed valid when it is able to fool the model (2) and when it is inside the L_p ball of a specified radius centered in $\mathbf{x}$ (1).

Recent literature has proposed attacks in the L_1, L_2, and L_∞ norm constraint scenarios. Perhaps the most common are L_∞ attacks, such as Moon et al. [21], which uses a surrogate problem to modify image patches, and [16] which computes universal patches to fool object detectors. Typically, L_1, L_2, and L_∞ attacks alter pixels by very small amounts, giving rise to human-imperceptible perturbations. However, by not constraining the number of perturbed pixels, these attacks often modify the large majority of the image (Fig. 1, L_1, L_2, L_∞ attacks), resulting in non-deployable perturbations. In contrast, attacks of L_0 nature such as Croce et al. [6], Narodytska and Kasiviswanathan [22] by definition limit the number of perturbed pixels. Thus, we choose to set Adversarial Scratches in the L_0-bounded scenario.

Perturbation structure: We categorize attacks on this axis based on specific constraints imposed on the perturbation, such as its localization or geometric structure (e.g., patches [31], objects [30], signatures [17]). As an example of a structured attack, we mention Patch Attack [31], which uses reinforcement learning to optimally

place pre-generated textured patches, albeit covering up to 20% of the entire image. Adversarial Scratches are structured as contiguous curves, this being a key element for the deployability of our attack. Indeed, as shown in Fig. 1, the perturbation of an unstructured attack, even of the L_0 type, cannot be contained in a spatially adjacent region in the image (C2).

Aim: We finally sort attacks into targeted ones, for which the goal is to force the prediction to a specific class y', such that $C(\mathbf{x}') = y'$, and untargeted ones, where the objective is simply to induce misclassification, independently of the resulting class, such that $C(\mathbf{x}') \neq y = C(\mathbf{x})$. For this categorization, we cite Sparse-RS [6], as it displays both targeted and untargeted attacks. In this work, we primarily focus on untargeted attacks. In Section 6.2, we discuss how to obtain targeted Adversarial Scratches.

In summary, Adversarial Scratches belong to the black-box (score-based) category, are L_0-bounded, and adopt the structure of deployable scratches. Alongside this formal categorization, we report in Table 1 a comparison of popular perturbation models in the literature, where we consider a variety of different factors, such as whether the attacks are L_0-bounded, were launched on a large, ImageNet-scale network, and most importantly whether these were deployable. The analysis shows strong similarities between our work and Patch-RS [6], the primary difference being the perturbation model. The attacks presented in Sparse-RS achieve state-of-the-art performance in several settings. In particular, Patch-RS is, to the best of our knowledge, the most effective black-box L_0 attack which also allows deployable perturbations, by modifying a single square patch in the image. As such, this is the most relevant method amongst those considered in our experiments. Nonetheless, we also compare to non-deployable attacks, since they represent an ideal reference amongst all L_0 attacks. The "any-pixel" attack from Croce et al. [6], which modifies any k pixels in the image, shows the best results amongst non-deployable L_0 attacks. We remark that comparing deployable attacks to non-deployable attacks is unfair, as the latter can exploit a much larger attack surface.

3. Methodology

We frame the generation of Adversarial Scratches as a constrained optimization problem. Given a trained CNN classifier f and an input image $\mathbf{x}$, the adversarial sample $\mathbf{x}'$ is found by minimizing the *margin loss*:

$$\mathcal{L}_f(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}')_y - \max_{i \neq y}(f(\mathbf{x}')_i) \tag{3}$$

subject to the L_0 bound:

$$\|\mathbf{x}' - \mathbf{x}\|_0 \leq k, \tag{4}$$

and to the localization constraint:

$$(\mathbf{r}[i, j] = 0) \Rightarrow (\mathbf{x}'[i, j] = \mathbf{x}[i, j]). \tag{5}$$

Table 2

Description of quadratic Bézier curve parametrization.

Parameter	Interpretation	Min value	Max value	Description
v_0	$P_{0,x}$	0	$w-1$	x coordinate of point P_0
v_1	$P_{0,y}$	0	$h-1$	y coordinate of point P_0
v_2	$P_{1,x}$	0	$w-1$	x coordinate of point P_1
v_3	$P_{1,y}$	0	$h-1$	y coordinate of point P_1
v_4	$P_{2,x}$	0	$w-1$	x coordinate of point P_2
v_5	$P_{2,y}$	0	$h-1$	y coordinate of point P_2
v_6	R	0	255	red component of scratch color*
v_7	G	0	255	green component of scratch color*
v_8	B	0	255	blue component of scratch color*

* Colors are restricted to be fully saturated, thus, the color components can only assume the extreme values 0 or 255 of the range. This means that there are only eight available colors: [0, 0, 0], [255, 0, 0], [0, 255, 0], [0, 0, 255], [255, 255, 0], [0, 255, 255], [255, 0, 255], and [255, 255, 255].
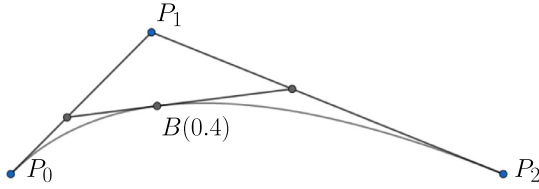

Fig. 4. Structure and construction scheme of a second-order Bézier curve $B(t)$ from its control points P_0, P_1, P_2. The curve is defined by interpolation of the three points by using a control parameter $t \in [0, 1]$. In the image, we depict an example location of the curve $B(0.4)$.

We define the margin loss (3) $\mathcal{L}_f$ as in Croce et al. [6] as the difference between the posterior probability $f(\mathbf{x}')_y$ of $\mathbf{x}'$ belonging to the ground truth class y, and the maximum posterior probability $\max_{i \neq y}(f(\mathbf{x}')_i)$ of $\mathbf{x}'$ belonging to a class other than y. The sample $\mathbf{x}'$ is misclassified when $\max_{i \neq y}(f(\mathbf{x}')_i) > f(\mathbf{x}')_y$.[1] Therefore, $\mathbf{x}'$ is an adversarial sample for $\mathbf{x}$ when:

$$\mathcal{L}_f(\mathbf{x}, \mathbf{x}') < 0. \tag{6}$$

We solve the constrained optimization problem through an iterative optimization procedure. The result is an image $\mathbf{x}^v$ obtained by superimposing a scratch B to the original sample $\mathbf{x}$, where B is modeled as a Bézier curve identified by parameter vector v. Fig. 3 visualizes how the perturbation is computed starting from this parameter configuration, highlighting the usage of region $\mathbf{r}$. In the remainder of the section, we detail the perturbation model and the search strategy that characterize Adversarial Scratches.

3.1. Perturbation model

The model we adopt to describe Adversarial Scratches is Bézier curves [13]. Bézier curves are polynomial segments, as such, they are continuous and continuously differentiable. While our solution is general and accounts for Bézier of any order, we illustrate our method for second-order curves, which have shown to perform best in our experimental evaluation. A second-order Bézier (Fig. 4) is defined as:

$$B(t) = (1-t)^2 P_0 + 2(1-t)t P_1 + t^2 P_2, \ 0 \leq t \leq 1. \tag{7}$$

As shown in Fig. 4, second-order Bézier curves are solely defined by their control points P_0, P_1, P_2, such that they start in P_0 and end in P_2, with point P_1 regulating their path. Thus, each scratch is identified by a vector $v \in \mathbb{R}^9$ (Table 2), where six parameters indicate the location of the Bézier's control points expressed in image coordinates, and three parameters indicate the R, G, B color components of the scratch. Following common practice, to reduce the size of the search space, we only consider extreme intensity values $\{0, 255\}$, meaning that only 8 RGB triplets are available.

We chose Bézier curves because, despite their compact parametric representation, they can express a wide range of shapes. The intuition behind their adversarial nature is that these curves introduce patterns in the image to which CNNs are sensitive. Crucially, Bézier curves possess the property of being arbitrarily subdividable (see Section 4.1). This allows to clip the perturbation to the target region, as shown in Fig. 3, and to bound them in L_0 terms. These properties, in turn, allow deployability of Adversarial Scratches.

Our experiments confirm that Adversarial Scratches are a powerful attack against CNN classifiers. Moreover, our perturbations can in principle be applied on a target by using a marker or a spray can, as opposed to patches which are only applicable through a printed sticker. Lastly, we note that while we have defined perturbations consisting of a single scratch, our formulation is more general. When more than one scratch is to be applied, the parameter vector v is expanded so that $v \in \mathbb{R}^{9n}$. In this case, the perturbation will be in the form of n Bézier curves, as discussed in Section 4.2.

3.2. Search strategy

The optimization problem described in Section 3.1 is solved by finding a parameter configuration v that leads to a successful adversarial attack. However, in the black-box attack scenario, the gradient of $\mathcal{L}_f$ is unknown. Therefore, search can only be performed by using gradient-free optimizers. Aside from being limited to this category of optimizers, the process is entirely transparent to the particular optimizer chosen to compute the perturbation.

We tested four popular gradient-free optimizers: random search (RS) [32], particle swarm optimization (PSO) [33], differential evolution (DE) [25], and neuro-genetic optimization (NGO) [27]. RS and DE were chosen as they were used in related works [6,10]. PSO and NGO were chosen as they represent evolutions of genetic algorithms such as DE. We discuss our choice of optimizer in Section 5.4.

3.3. Attack procedure

Algorithm 1 describes how the attack is performed on an image sample. For clarity, we only describe the application of a single scratch. Section 4.2 details the extension of this procedure to multiple scratches. The required inputs are: the image $\mathbf{x}$ which we want to perturb, its target class y, the query threshold *MAX_ITER*, the model f which we want to attack, the *optimizer* that will solve the optimization problem (3), the L_0 bound k, and the region $\mathbf{r}$ used to restrict the attack only to the pixels belonging to the target object.

---

[1] This holds also when $\mathbf{x}$ is not correctly classified ($C(\mathbf{x}) \neq y$), and in this case we have $\mathcal{L}_f(\mathbf{x}, \mathbf{x}) < 0$. However, $\mathbf{x}$ can never be considered as an adversarial sample for itself. For this reason, in our experiments, we follow the common practice of discarding all misclassified images $\mathbf{x}$.

Algorithm 1 General attack procedure.

Require: $\mathbf{x}$, y, *MAX_ITER*, f, *optimizer*, k, $\mathbf{r}$

1: $iter \leftarrow 0$
2: $C \leftarrow argmax_i f(\mathbf{x})_i$ ▷ Predict class for original image
3: **for** $i = 1$ to *MAX_ITER* **do**
4:     $v \leftarrow optimizer.ask()$ ▷ Obtain a solution from the optimizer
5:     $v^* \leftarrow clip(v, k, \mathbf{r})$ ▷ Clip thescratch (Algorithm 2)
6:     $B^* \leftarrow Bezier(v_0^*, \ldots, v_5^*)$ ▷ Sample Bézier
7:     $c \leftarrow (v_6^*, v_7^*, v_8^*)$
        ▷ Apply scratch to obtain adversarial image x^{v^*}
8:     $\mathbf{x}^{v^*} \leftarrow \mathbf{x}$
9:     **for** $j = 1$ to $|B^*|$ **do**
10:         $\mathbf{x}^{v^*}[b_j^*] \leftarrow c$ ▷ Set pixels on the scratch support to the scratch color
11:     **end for**
12:     $l \leftarrow f_y(\mathbf{x}^{v^*}) - \max_{z \neq y} f_z(\mathbf{x}^{v^*})$ ▷ Feed $\mathbf{x}^{v^*}$ to the model and compute loss
13:     **if** $l < 0$ **then**
14:         $iter \leftarrow i$
15:         **break**
16:     **end if**
17:     $optimizer.tell(v, l)$ ▷ Update the optimizer information
18: **end for**
19: **if** $iter > 0$ **then**
20:     **successful attack in [iter] iterations**
21: **else**
22:     **unsuccessful attack**
23: **end if**

Algorithm 2 Scratch clipping.

Require: v, k, $\mathbf{r}$

1: $P_0 = \{v_0, v_1\}, P_1 = \{v_2, v_3\}, P_2 = \{v_4, v_5\}$ ▷ Input Bézier control points
2: $flag \leftarrow 0, d \leftarrow 0, B^* \leftarrow \emptyset$ ▷ Initialize flags and scratch support
    ▷ Sample the Bézier to obtain points and their parametric position
3: $B, T \leftarrow Bezier(v_0, \ldots, v_5)$
4: **for** $j = 1$ to $|B|$ **do**
5:     **if** $flag == 0$ and $\mathbf{r}[B] == 1$ **then**   ▷ Beginning of the segment found
6:         $p \leftarrow j$
7:         $flag \leftarrow 1$
8:     **end if**
9:     **if** $flag == 1$ **then**
10:         **if** $\mathbf{r}[B] == 0$ or $d == k$ **then**   ▷ End of the segment is reached
11:             $q \leftarrow d - 1$
12:             **break**
13:         **else**
14:             d++
15:             $B^* \leftarrow B^* \bigcup b_j$ ▷ Add the location to the support
16:         **end if**
17:     **end if**
18: **end for**
    ▷ Split Bézier and return new parameter configuration
19: $P_0^* \leftarrow (b_{1,x}^*, b_{1,y}^*)$
20: $P_1' \leftarrow (1 - t_p)P_1 + t_p P_2$
21: $P_1'' \leftarrow (1 - t_{p+q})b_p + t_{p+q}P_1'$
22: $P_2^* \leftarrow (b_{|B^*|,x}^*, b_{|B^*|,y}^*)$
23: $v \leftarrow \{P_{0,x}^*, P_{0,y}^*, P_{1,x}'', P_{1,y}'', P_{2,x}^*, P_{2,y}^*, v_6, v_7, v_8\}$
24: **return** v^*

In the first phase, the iteration count is initialized and the model's prediction for the original sample is computed (Lines 1–2). Then, the algorithm proceeds with an iterative procedure that loops until either a valid adversarial sample is found or when a query limit is reached (Lines 3–18). Within this loop, the first step is to request a solution from the optimizer through the standard ask-and-tell interface. By calling *optimizer.ask* (Line 4), the optimizer produces a candidate solution in the form of a vector v, which is detailed in Table 2 for second-order Bézier curves. Because the optimizer is not constrained to satisfy (4) and (5), v could represent a scratch which doesn't meet the L_0 bound or the localization constraints given by the region $\mathbf{r}$. To satisfy the above constraints, the *clip* operation (Line 5) returns a second parameter configuration v^* that corresponds to a scratch which is a subset of the original one. This is possible since the set of Bézier curves is closed with respect to arbitrary subdivision [1]. The details of the clipping and parameter update procedure are described in Algorithm 2. Given v^*, we compute *i)* the support set of the scratch $B^* = \{(x_1, y_1), \ldots\}$ by sampling curve (7) and *ii)* its color $c = \{v_6^*, v_7^*, v_8^*\}$ (Lines 6–7). The tentative adversarial image $\mathbf{x}^{v^*}$ is then obtained by changing the color of $\mathbf{x}$ for each pixel location in the scratch support B^* to the scratch color c (Lines 8–11):

$$\mathbf{x}^{v^*}[i, j] = \begin{cases} c & \text{if } (i, j) \in B^* \\ \mathbf{x}[i, j] & \text{otherwise} \end{cases}. \tag{8}$$

Thus, we compute the margin loss (3) by feeding the tentative adversarial image $\mathbf{x}^{v^*}$ to the target model f (Line 12), and if the attack is successful (6), the procedure ends (Line 20). Otherwise, the state of the optimizer is updated (Line 17) by providing the loss value for the given configuration, and the procedure continues to the next iteration. If no adversarial sample can be computed within the query limit, the attack is unsuccessful and the algorithm terminates (Line 22). In our experiments, in line with [6], the query limit was set to 10,000. This is better discussed in Section 5.1.

We note that the clipping operation transforms the parameter vector v before the sample is perturbed. Thus, when updating the state of the optimizer, the parameter configuration v is associated to the margin loss $\mathcal{L}_f(\mathbf{x}, \mathbf{x}^{v^*})$ where the argument is the sample with the clipped perturbation $\mathbf{x}^{v^*}$ instead of $\mathbf{x}^v$. This choice is justified since clipping does not modify feasible solutions, therefore, for all feasible $\mathbf{x}^v$, $\mathcal{L}_f(\mathbf{x}, \mathbf{x}^v) = \mathcal{L}_f(\mathbf{x}, \mathbf{x}^{v^*})$.

4. Implementation details

In this section, we provide practical details regarding the generation of adversarial examples as shown in Algorithm 1, and discuss the extension of the attack to multiple scratches.

4.1. Scratch clipping

Since the optimizer is agnostic to the underlying perturbation model, it may propose parameters v that may represent scratches that do not satisfy the deployability constraints. We address this issue by clipping the scratch to a connected segment of the original one, such that the clipped scratch satisfies the L_0 constraint and is entirely contained in the target region. The clipping procedure, described in Algorithm 2, receives as input the configuration v, the L_0 bound k, and the region $\mathbf{r}$, and returns a new parameter configuration v^*. This procedure modifies the coordinates of the control points of the Bézier, leaving its color unchanged. For second-order Bézier curves (Table 2), this means modifying parameters $v_0, \ldots, v_5$.

In practice, we sample the scratch identified by v and obtain its support B (Lines 1–3). Then (Lines 4–18), we select the longest contiguous subset $B^* = \{b_p, b_{p+1}, \ldots, b_{p+q}\}$ of B which is entirely

contained in the region **r**, starting at the first pixel[2] of B belonging to the mask. The set of all Bézier curves is closed with respect to arbitrary subdivision [1], thus, given any two points P_0^*, P_2^* along a Bézier B, it is possible to compute the parameters of a new Bézier B^* which is a segment of B starting in P_0^* and ending in P_2^*. We achieve this by using the arbitrary subdivision procedure [1] up to two times (Lines 19−20). By the first subdivision, we remove the segment $\{b_1, \ldots, b_p\}$, obtaining a Bézier starting in P_0^*, ending in P_2, and having control point P_1'. By the second subdivision we remove $\{b_{p+q} + 1, \ldots, b_{|B|}\}$, finally obtaining a Bézier starting in P_0^*, ending in P_1^*, having control point P_1''. Lastly (Lines 21−22), we return the new parameter configuration v^* representing the clipped scratch. By construction, the new Bézier satisfies the L_0 and localization constraints.

4.2. Multiple scratches

Algorithm 1 describes how to apply a single Adversarial Scratch. We may however want to perturb **x** by using n scratches. For second-order Bézier curves, the parameter vector describing n scratches will be $v \in \mathbb{R}^{9n}$, where each tuple of 9 parameters represents one scratch, having support B_s and color $c_s, s = 1 \ldots n$. Before application, each scratch needs to be clipped according to Algorithm 2 to obtain the clipped supports B_s^* and parameter configuration v^*, then, the scratches are applied in sequence, possibly overwriting already perturbed pixels when these overlap. We extend (8) for the case with $n > 1$ scratches, so that the image where scratches $1 \ldots s$ are applied is defined as follows:

$$\mathbf{x}^{(s)}[i, j] = \begin{cases} c_s & \text{if } (i, j) \in B_s^* \\ \mathbf{x}^{(s-1)}[i, j] & \text{otherwise} \end{cases}, \tag{9}$$

where $\mathbf{x}^{(0)} = \mathbf{x}$ and the end result is $\mathbf{x}^{v^*} = \mathbf{x}^{(n)}$.

Increasing the number of scratches makes the attack more powerful, as it allows the perturbation to cover different regions within **r**, but comes at the cost of deployability, as one would need to account for the relative positions between scratches. In practice, we limit to a maximum of $n = 5$ Adversarial Scratches, which we deem to be a reasonable bound allowing for powerful yet deployable attacks. In Section 5.4 we study the effects of using attacks with varying number of scratches.

5. Experiments

In this section, we analyze the performance of Adversarial Scratches in a variety of experiments, utilizing the ImageNet [28] and TSRD [23] datasets. First, we describe the employed experimental setup (Section 5.1), then, we compare Adversarial Scratches to state-of-the-art deployable attacks, presenting experiments on the ImageNet and TSRD datasets, and against Microsoft Cognitive Services API (Section 5.2). We then compare Adversarial Scratches to non-deployable state-of-the-art attacks (Section 5.3), as they represent an ideal performance reference for L_0 attacks. Lastly, we present a thorough exploration of several possible configurations of Adversarial Scratches (Section 5.4). All experiments target ResNet-50 classifiers, except where otherwise stated. Since many considered attacks have a stochastic component, we average results from five runs with different random seeds, also reporting standard deviations across runs.

5.1. Experimental setup

We introduce our experimental evaluation framework, and detail the figures of merit used throughout the experiments.

5.1.1. Framework

We have developed a flexible Python framework to run our experiments. The framework allows to design, implement, and execute adversarial attacks on images. The model to be attacked, the optimizer, and the perturbation model are modular, and can be combined to perform a variety of tests in different combinations. The modules communicate via straightforward interfaces, allowing seamless integration and extension to many attack methodologies, including those implemented in Sparse-RS [6]. We release our code to the public as a platform to test adversarial attacks on neural networks.

5.1.2. Metrics

To measure the performance of the attacks, we compute fooling rate (FR), average queries (AQ), and median queries (MQ). These metrics are significantly influenced by the query limit, thus, following common practice, all methods are compared using the same query limit of 10,000. The metrics are defined as follows:

1. **Fooling Rate (FR)** is the fraction of image samples for which the attack was successful within the query limit, out of all samples that were subject to the attack. Higher FR indicates that the attack was more successful.
2. **Average Queries (AQ)** is the average number of queries needed to craft a successful perturbation. Lower AQ values indicate that the attack needs fewer attempts, on average, to find a perturbation that deceives the model.[3]
3. **Median Queries (MQ)** is the median number of queries needed to craft a successful perturbation. MQ is useful since query requirements may greatly vary between samples. Therefore, AQ and MQ, in conjunction, allow to better understand the distribution of query requirements.[3]

5.1.3. Considered methods

In our experiments, we compare Adversarial Scratches to several state-of-the-art L_0 attacks. Following [7] and [20], we do not consider L_1, L_2, and L_∞ attacks, since given a specific L_0 bound it is impossible to define L_1, L_2, and L_∞ bounds that would result in comparable perturbations. This is further justified by our interest in deployable attacks. Indeed, as stated in Section 1, L_1, L_2, and L_∞ attacks cannot be deployed.

5.1.4. Optimizer

Results for Adversarial Scratches have been obtained using the NGO optimizer, as it shows the best performance amongst the considered ones. In Section 5.4 we compare the performance of various optimizers.

5.2. Comparison against deployable attacks

We analyze the performance of Adversarial Scratches against state-of-the-art deployable attacks on the widely used ImageNet dataset. To further investigate the deployability of our attacks, we perform experiments on the TSRD traffic sign dataset and we deploy our attack against the publicly available Microsoft Cognitive Services API.

5.2.1. Considered methods

We compare against Patch-RS [6], PatchAttack [31], and LOAP w/ GE [6,26], as these are the best performing deployable attacks in the literature. All of these attacks are based on a perturbation model that modifies a square patch (Fig. 5, left), differing only

---

[2] The ordering is given by the parametric location t along the Bézier curve (7)

[3] Both AQ and MQ are computed over images where the attack was successful within the query limit.

Patch   Adversarial Scratches   "any-pixel"

$L_0 = 400$

☐ Deployable
☐ Non-deployable

The attack perturbs a square region of size $\sqrt{k}$ pixels

The attack places three scratches, one pixel wide and $\frac{k}{3}$ pixels long
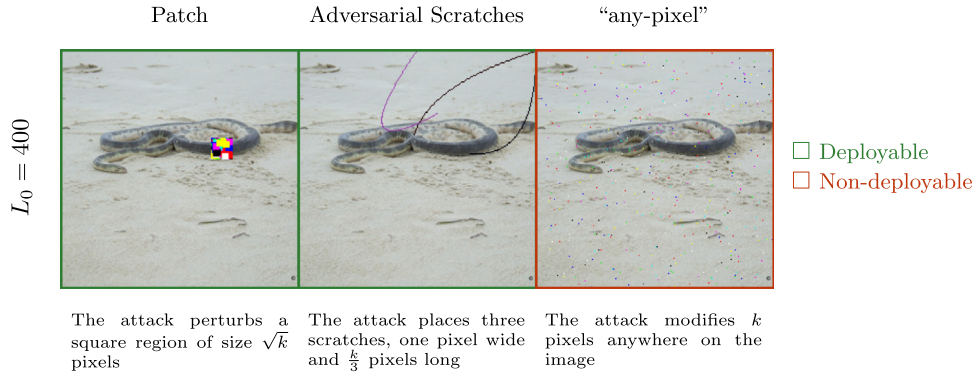
The attack modifies k pixels anywhere on the image

Fig. 5. Different attack structures with the same L_0 constraint of 400 perturbed pixels. On the left, a patch attack [6,26,31], in the middle, our 3-scratch attack, and on the right, the "any-pixel" attack [6].

in how the patch is defined and optimized. Patch-RS uses random search to overlap colored rectangles within the patch area, while PatchAttack uses an RL agent to optimize a textured adversarial patch. Lastly, LOAP is originally a white-box method that optimizes the adversarial patch through gradient information, which is adapted to the black-box scenario by using Gradient Estimation, as in Croce et al. [6], resulting in the black-box LOAP w/ GE.

5.2.2. Experiments on ImageNet

This experiment provides a comparison to the state-of-the-art in a standard setting with no restrictions on the target region.

We compare 20×20 patches generated with Patch-RS, LOAP w/ GE, and PatchAttack (total $L_0 = 400$) to an attack composed of three $L_0 = 133$ scratches. L_0 bounds are thus comparable, as this Adversarial Scratches attack is constrained to a total $L_0 = 399$ (Fig. 5, center). Since our experimental setup is equivalent to that of Sparse-RS [6], results for Patch-RS, LOAP w/ GE, and PatchAttack are taken as reported from their paper.

Table 3 shows that Adversarial Scratches have much higher fooling rate while requiring significantly fewer queries than other compared methods. Moreover, we note that Adversarial Scratches are typically shorter than their nominal length. Indeed, in this experiment, Adversarial Scratches modify on average 331.9 pixels, fewer than the allowed 399 pixels. Other attacks, in contrast, always modify all 400 allowed pixels. These results show that Adversarial Scratches are better than current state-of-the-art deployable attacks in all considered metrics.

5.2.3. Experiments on TSRD

To better tests the deployability of the attacks, we design an experiment on the TSRD [23] dataset where attacks are only allowed to perturb pixels in a target region which corresponds to a traffic sign. The target model is a ResNet-50 classifier finetuned to 98% test accuracy on the TSRD dataset.

Considered methods: For this experiment, we only compare to Patch-RS as it was shown (Table 3) to be the best performing deployable attack amongst the tested ones. However, in its base form, Patch-RS cannot be localized within a target region. To solve this issue, we have developed a spatially localizable version of Patch-RS, namely, R-Patch-RS. This was achieved with minimal modifica-

tions to Sparse-RS' code. Fig. 6 shows an example of Adversarial Scratches (center) and R-Patch-RS (left) attacks on TSRD images.

Dataset preparation and model setup: The TSRD [23] dataset is composed of 6164 traffic sign images, divided in 58 classes. For this experiment, we have created target regions by manually segmenting non-occluded pixels of traffic signs from more than 100 TSRD samples. We remark that this manual annotation was only performed to conform to a realistic scenario where the attack perturbation is limited to the region described by the street sign.

Since the TSRD dataset includes augmented and duplicate images, to favor unique images, the images used for this experiment were selected manually. Selected images and their augmented versions were never used for training, ensuring that attacks are performed on images the model has never seen. We publicly release these segmentation masks, together with information on how to match them with images in the original TSRD dataset.

Results: Table 4 shows that, although the attacks are constrained to only affect pixels belonging to the traffic sign in the image, both Adversarial Scratches and R-Patch-RS are very successful, and ultimately results in almost all samples being successfully attacked. We find that Adversarial Scratches have better AQ and MQ than R-Patch-RS. Most notably, median queries drop from 195 to 6, meaning that, by using Adversarial Scratches, we could attack half the traffic signs in the dataset with just 6 attempts.

5.2.4. Experiments on microsoft cognitive services API

We perform an attack against the Microsoft Cognitive Services Image Captioning API [19] using Adversarial Scratches. We formalize this API as a model g which, given an image $\mathbf{x}$, provides a caption describing the image's content and the model's confidence $h(\mathbf{x})$. Since the margin loss (3) cannot be computed in the captioning scenario, we solve the optimization problem by minimizing the loss:

$$\mathcal{L}_g = h(\mathbf{x}). \tag{10}$$

The rationale behind this is to minimize the confidence to induce the model to produce wrong captions. In this experiment, we analyze the performance of Adversarial Scratches against a real-world system, however, since the target is an online service, physical deployability is irrelevant. Thus, in this setting, we use 3 scratches

Table 3

Comparison to deployable attacks. Results for Patch-RS, LOAP w/ GE and PatchAttack are reported from Croce et al. [6], which uses the same experimental setup. Results are computed using a query limit of 10,000.

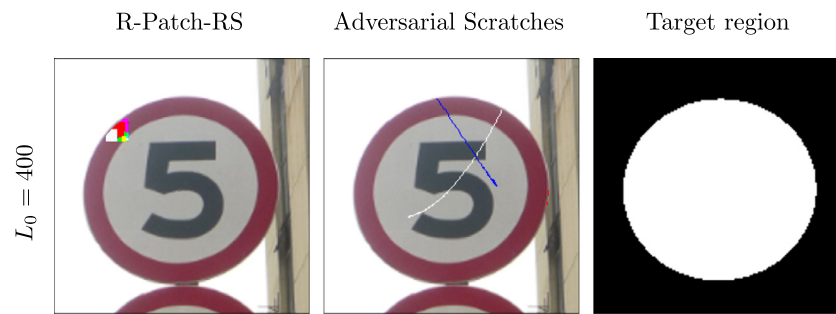| Attack	FR	AQ	MQ	Perturbation model
LOAP w/ GE	40.6% ±0.1%	6870 ±10	10,000 ±0	20 × 20 patch
PatchAttack	49.6% ±1.2%	5722 ±64	5280 ±593	20 × 20 patch
Patch-RS	79.5% ±1.4%	2808 ±89	438 ±68	20 × 20 patch
Adversarial Scratches	**97.9% ± 0.3%**	**302 ± 38**	**27 ± 3**	Three 133px long Bézier

Fig. 6. Deployable attacks applied to the TSRD street sign dataset. Left: a 20×20 R-Patch-RS attack. Centre: an attack composed of two Adversarial Scratches with L_0 bound set to 200. Both attacks are constrained to the street sign's region **r**, which is depicted on the right.
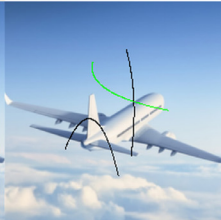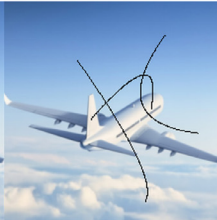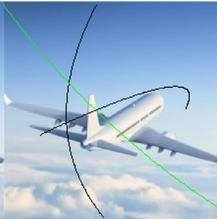

Fig. 7. Successful attacks against the Microsoft Cognitive Services Image Captioning API. The top left image denotes the source image, which is captioned correctly as 'a plane flying in the sky'. All other images are examples of images that successfully deceived the captioning service. Note that, even though each of the three scratches composing the attack was bounded to $L_0 \leq 500$, the overall L_0 difference with respect to the original is much lower than 1500.

each with $L_0 \leq 500$, with no restriction to a target region. As API calls are rate limited, it is unfeasible to perform thousands of consecutive queries, thus, we have attacked a single image and provide qualitative results in Fig. 7.

Results: Adversarial Scratches were able to deceive the API into generating wrong captions. Most notably, we were able to significantly change the output caption in just 6 iterations (the caption was altered also in earlier iterations but not significantly, e.g. "A plane" was substituted by "A jet"). Examples of these gen-

erated scratches are shown in Fig. 7. The attacks were performed on `September 6th, 2021`. *The vulnerability was reported to Microsoft.*

5.3. Comparison against non-deployable attacks

We compare the performance of Adversarial Scratches to that of other non-deployable L_0 attacks. Importantly, *this comparison is unfair to Adversarial Scratches*, since the non-deployable attacks we

Table 4
Performance of attacks on the TSRD dataset when restricted to the target's pixels.

Attack	FR	AQ	MQ	Comment
R-Patch-RS	97.5% ±1.4%	366 ±58	195 ±15	20×20 patch ($L_0 = 400$)
Adversarial Scratches	**100**% ± **0.0**%	**68 ± 31**	**6 ± 1**	Three Bézier each 133px long ($L_0 = 399$)

Table 5

Comparison to non-deployable L_0 attacks. SimBA is only run once as the provided code does not allow changing seed.

		Attack	FR	AQ	MQ	Perturbation model
L_0	400	SimBA	71.3%	500.15	457	Any 1200 pixel channels in the image
		Adversarial Scratches	97.9% ±0.3%	302 ±38	27 ±3	Three Bézier, each 133px long
		"any-pixel"	**99.9**% ± **0.2**%	**154 ± 6**	**25 ± 1**	Any 400 pixels in the image
	50	SimBA	12.1%	72.6	68	Any 150 pixel channels in the image
		Adversarial Scratches	55.8% ±0.4%	**866 ± 35**	**75 ± 7**	One Bézier, 50px long
		"any-pixel"	**83.9**% ± **0.8**%	1899 ±48	906 ±23	Any 400 pixels in the image

Table 6

Performance of different optimizers for the $L_0 = 400$ attack for three Bézier line.

Optimizer	FR	Comment
RS	83.6% ±0.5%	Random search with step size scheduling [6]
PSO	90.5% ±0.6%	Implementation provided by NeverGrad [27]
DE	93.9% ±0.9%	Population size 20 and restarts each 200 iterations
NGO	**97.9**% ± **0.3**%	Implementation provided by NeverGrad [27]

Table 7

Performance of $L_0 = 400$ Bézier attacks with different number of scratches.

Bézier count	Per-Bézier L_0	FR	AQ	MQ	Average L_0
1	400	89.6% ±0.3%	509 ±71	54 ±2	**186.7 ± 1.7**
2	200	96.6% ±0.4%	316 ±39	37 ±2	280.1.7 ±0.7
3	133	**97.9**% ± **0.3**%	302 ±38	27 ±3	331.9 ±1.1
4	100	**97.9**% ± **0.1**%	**281 ± 18**	24 ±1	359.9 ±0.6
5	80	**97.9**% ± **0.0**%	301 ±11	**23 ± 1**	373.2 ±1.3

compare to are more general than Adversarial Scratches and can exploit a much wider attack surface.

5.3.1. Considered methods

We compare to the "any-pixel" attack from Sparse-RS (Fig. 5, right), which uses random search to optimize a perturbation of any k pixels in the image, and SimBA [11], which finds orthonormal directions to iteratively improve the perturbation. We chose SimBA and Sparse-RS's "any-pixel" attack as they are state-of-the-art attacks in the black-box, L_0 scenario.

5.3.2. Experimental setup

Since we are comparing to non-deployable attacks, we only focus on the ImageNet dataset, with no restrictions on the target region. We test under L_0 constraints of 400 and 50 pixels. In the $L_0 = 400$ scenario, we use an attack composed of three $L_0 = 133$ scratches. For the $L_0 = 50$ scenario, we use a single $L_0 = 50$ scratch, as shorter segments would not display features typical of scratches. All attacks are run with query limit set to 10,000, exception made for SimBA, as this attack perturbs one color channel of one pixel each iteration. To enable a fair comparison, SimBA was limited to a number of iterations equal to three times the L_0 bound. Although this results in a lower query limit than 10,000, this goes in favour of SimBA as the attack can potentially modify three times more pixels.

5.3.3. Results

Table 5 shows that, in the $L_0 = 400$ scenario, the fooling rate of Adversarial Scratches is comparable to the state-of-the-art, with Sparse-RS' "any-pixel" attack being marginally better, and SimBA being worse than both. The outcome is different in the very strict $L_0 = 50$ case, where the non deployable Sparse-RS is still able to achieve 83.9% FR, while our attack's performance drops significantly to 55.8%. The performance of SimBA is also greatly reduced, demonstrating that this is an extremely constrained test scenario. These results show that, when Adversarial Scratches can assume their typical shapes, as is the case for the $L_0 = 400$ scenario, our

attack achieves surprisingly good performance, comparable even to those of non-deployable state-of-the-art attacks. This is not true for the $L_0 = 50$ scenario, where scratches are too short to display adversarial features.

5.4. Exploration of parametric configuration of adversarial scratches

We explore several configurations of Adversarial Scratches, in terms of search strategy, number of scratches, order of the Bézier curve, and color configuration. All the experiments in this section are performed on 1000 samples of the ImageNet dataset.

5.4.1. Search strategy

As discussed in Section 3.2, we have tested four optimizers: Differential Evolution (DE) [25], Particle Swarm Optimization (PSO) [33], Neuro-Genetic Optimization (NGO) [27] and Random Search (RS) [32]. We focus on the $L_0 = 400$ case, and test an attack composed of three $L_0 = 133$ scratches. We test the RS optimizer using our own implementation of Sparse-RS' Random Search algorithm with scheduling. We also modify the standard DE implementation to return solutions which are within the search boundaries of Table 2. Lastly, we use default PSO and NGO implementations from Nevergrad [27]. Table 6 shows high fooling rates for all tested optimizers, with NGO displaying the best performance. Remarkably, these results demonstrate that Adversarial Scratches are effective across a variety of optimizers, which may be due to the small dimensionality of the parameter space required by the attack.

5.4.2. Scratch quantity

The number of scratches is a parameter that significantly influences deployability. Attacks with larger number of scratches may be more powerful, since they can cover a larger region in the image. This, however, comes at the cost of deployability, as the relative position between scratches must be accounted for. Example attacks with varying number of scratches are displayed in Fig. 8.
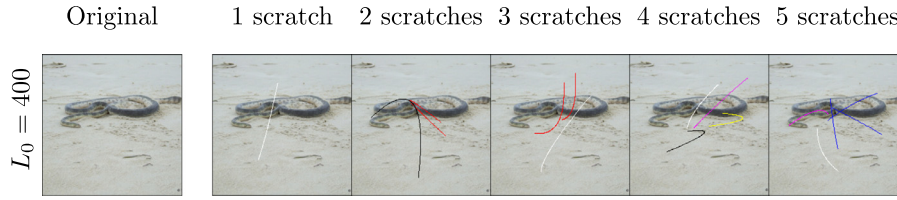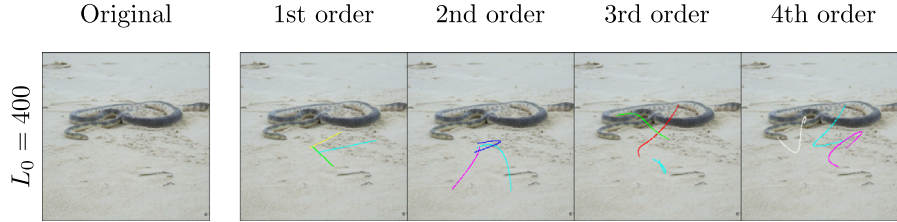
Fig. 8. $L_0 = 400$ attacks with different number of scratches.

Fig. 9. Example attacks with Bézier of varying order (three Bézier, each with $L_0 = 133$).

Table 7 shows that *FR* improves when increasing the number of scratches from 1 to 3, but there is no difference when further increasing from 3 to 5. *MQ* decreases as the number of scratches increases, indicating a higher chance of finding a solution in the very first iterations. However, as indicated by the increase in *AQ*, search becomes more challenging as the dimensionality of the parameter space increases. This result supports the adoption of three scratches for the attacks discussed in Section 5.2. From the "Average L_0" column, we also deduce that using several, shorter scratches results in perturbations which are closer to the L_0 limit.

5.4.3. Order of Bézier

We test Adversarial Scratches modeled as Bézier curves of varying order, as shown in Fig. 9. Higher order curves may express more complex shapes, and in turn lead to better fooling rates. Such expressive power, however, comes at the cost of parametrization efficiency and deployability, since more control points need to be defined and resulting curves may be harder to draw. The formulation (7) can be generalized to express a Bézier of order n as follows:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i, \ 0 \leq t \leq 1, \tag{11}$$

where $P_i, i = 0, \ldots, n$ are $n+1$ control points in the form (x_i, y_i). The degenerate case $n = 1$ gives straight line segments, while for $n > 2$ the curve may intersect itself. The parametrization of a Bézier of order n requires $2(n+1) + 3$ parameters, since there are $n+1$ control points and 3 color components.

We run the attack using $L_0 = 133$ Bézier curves testing Bézier order 1 to 4. Table 8 shows that Adversarial Scratches achieve similar performance across all tested Bézier orders, with order 2 having marginal better *FR*, order 4 better *AQ*, and order 3 better *MQ*. Our choice of using second-order Bézier curves in the experiments of Sections 5.2 and 5.3 is thus motivated, since they have

Table 8
Performance of $L_0 = 400$ Bézier attacks with varying order.

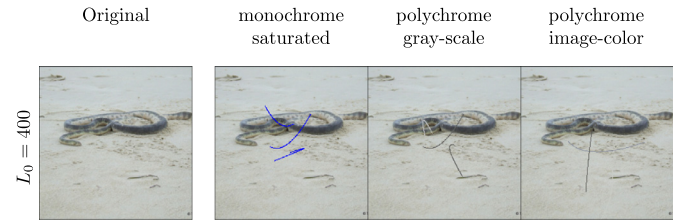| Bézier order	FR	AQ	MQ
1	96.3% ±0.1%	302 ±33	31 ±3
2	**97.9**% ± **0.3**%	302 ±38	27 ±3
3	97.6% ±0.2%	290 ±24	**25 ± 3**
4	97.3% ±0.5%	**265 ± 26**	28 ±2

Fig. 10. Example attacks with different color parametrization (three Bézier, each with $L_0 = 133$).

Table 9
Performance of $L_0 = 400$ Bézier attacks with varying color configuration.

Color configuration	FR	AQ	MQ
polychrome, saturated	**97.9**% ± **0.3**%	302 ±38	27 ±3
monochrome, saturated	97.6% ±0.1%	**252 ± 24**	**24** ±2
polychrome, gray-scale	92.4% ±0.6%	534 ±27	94 ±4
polychrome, image-color	87.7% ±0.6%	757 ±66	128 ±9

optimal performance and are more deployable than higher-order ones.

5.4.4. Color configuration

We test different color configurations and analyze changes in attack performance. Focusing on an attack composed of three quadratic $L_0 = 133$ Bézier curves, we test:

- A "polychrome, saturated" attack where each scratch assumes one of eight fully saturated colors;
- A "monochrome, saturated" attack where all scratches have the same fully saturated color;
- A "polychrome, gray-scale" attack where each scratch assumes one gray-scale color;
- A "polychrome, image-color" attack where each scratch assumes color equal to one of the available pixels in the image.

Fig. 10 displays samples perturbed with these color configurations. Table 9 shows that the "monochrome, saturated" attack has slightly better performance in terms of AQ and MQ than the baseline attack with three independently colored scratches, while still matching it in terms of FR. The gray-scale and image-color attack also show good performance, albeit lower than that of the other attacks, as less saturated colors have diminished attacking power.

Table 10
Recovery Rate and effects of the defenses on model performance.

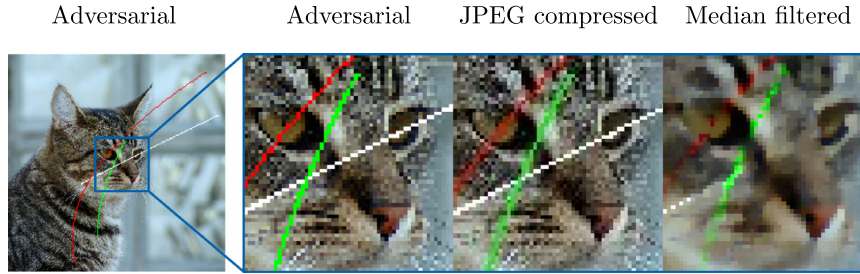| Defense strategy	Adversarial samples	Original images	
	Recovery Rate	Model performance	Performance delta
No defense (baseline)	*0%*	**76.7%**	**0%**
Median filtering, 3×3 kernel	**77.1**%	73.0%	−3.7%
JPEG, quality = 85	41.4%	74.4%	−2.3%
JPEG, quality = 90	39.0%	74.9%	−1.8%
JPEG, quality = 95	32.6%	75.7%	−1.0%
JPEG, quality = 99	25.7%	76.1%	−0.6%

Fig. 11. Defenses to Adversarial Scratches. We visualize the results of two defense mechanisms against three Adversarial Scratches, each with L_0 bound 133. The defenses are JPEG compression and median filtering. Median filtering is more effective as it removes most of the scratches from the image, while JPEG compression only marginally alters the perturbation.

6. Discussion

Our experiments show that Adversarial Scratches outperform other state-of-the-art deployable attacks, achieving comparable performance even against non-deployable L_0 black-box attacks. Furthermore, our experiments on the TSRD dataset show promising results for the applicability of Adversarial Scratches to physical targets. We attribute the success of Adversarial Scratches to the greatly reduced search space compared to other attacks, especially those presented in Sparse-RS. Indeed, the "any-pixel" attack has a search space with dimensionality $5k$, where k is the L_0 bound. Adversarial Scratches, instead, only require 9 parameters for each scratch, independently of the L_0 bound. In our experiments, we used single scratches for the $L_0 = 50$ case, which means a reduction of parameter count of more than $27\times$ (9 parameters for a single scratch attack against 250 parameters for $L_0 = 50$ "any-pixel" attack). For the $L_0 = 400$, we used 3 scratches, resulting in 74 times fewer parameters (27 parameters for three scratches against 2000 parameters for $L_0 = 400$ "any-pixel" attack).

6.1. Defenses

Recent works have proposed methodologies to counteract the effects of adversarial perturbations. These techniques encompass robust training procedures [18], perturbation detection and removal [12], and reconstructions through deep image priors [8]. To defend from Adversarial Scratches, following [9], we consider defenses that rely on *input filtering*, as these are more scalable than defenses that try to make the model itself more robust. In particular, we adopt:

- **JPEG compression** with varying quality factors.
- **Median filtering** separately to each channel with a kernel size of 3×3 pixels.

Metrics: We assess the effectiveness of defense $d(\cdot)$ through the *recovery rate* (*RR*), defined as the fraction of successful adversarial samples whose filtered version is correctly classified:

$$RR = \frac{|\{C(\mathbf{x}) = y\} \bigcap \{C(\mathbf{x}') \neq y\} \bigcap \{C(d(\mathbf{x}')) = y\}|}{|\{C(\mathbf{x}) = y\} \bigcap \{C(\mathbf{x}') \neq y\}|}, \quad (12)$$

where $X' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_n\}$ is the set of perturbed images (possibly including those for which the attack was not successful) for samples $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.[4]

Results: We assess the effectiveness of these defenses against a "polychrome, saturated" attack, using three $L_0 = 133$ second-order Bézier curves. The attack achieved 97.9% *FR* on 1000 samples of ImageNet, therefore, our defense analysis is based on a large pool of 979 adversarial samples, from which we exclude originally misclassified images (12).

Table 10 shows that both JPEG compression and median filtering are effective defenses, with median filtering providing the highest recovery rate (77.1%). Fig. 11 shows that median filtering can indeed remove scratch pixels from the image. JPEG compression is also a viable technique to recover the original target class (up to 41.4% recovery rate with quality = 85).

A drawback of filtering-based defenses is that these transformations may result in a drop in classification performance on non-adversarial images. To study this phenomenon, we compute the performance of the CNN used for our experiments on 1000 samples from the ImageNet validation set. Then, we apply the defense to each image in this set, and compute the performance drop on the filtered samples. Table 10 shows that the most effective median filtering is also the most detrimental to the model's performance, resulting in a drop in accuracy of 3.7%. We argue, however, that the high recovery rate justifies this minor performance drop.

6.2. Targeted attacks

In our work, we have focused on untargeted attacks. Nonetheless, Adversarial Scratches are easily extendable to the targeted scenario. As detailed in Section 2.2, targeted attacks generate adversarial samples $\mathbf{x}'$ that are classified as belonging to a specific target class y', namely $C(\mathbf{x}') = y'$. To obtain targeted Adversarial Scratches, we replace the optimization objective (3) with a cross entropy loss $\mathcal{H}$, where the target vector is all zeroes except for the

---

[4] If the filtered image is classified differently than the adversarial one $C(\mathbf{x}') \neq C(d(\mathbf{x}))$, but the resulting class is not the target $C(d(\mathbf{x})) \neq y$, the image is not considered to be recovered.

element in position y':

$$\mathcal{H}(f(x), [0, \ldots, 0, 1, 0, \ldots, 0]) = -\log(f(x)_{y'}) - \sum_{i \neq y'} \log(1 - f(x)_i). \tag{13}$$

In the untargeted scenario, the lower bound $\mathcal{L}_f(\mathbf{x}, \mathbf{x}')$ (6) identifies the threshold below which an image is adversarial. This is not the case for targeted attacks, where classification (and thus attack success) must be explicitly verified by checking whether $C(\mathbf{x}') = y'$.

7. Conclusions and future works

In this paper, we propose *Adversarial Scratches*: a novel attack structured as parametric Bézier curves applied to the image, and designed to be *deployable* to physical targets. We believe that our study of adversarial attacks is very relevant to the security of applications making use of deep learning models, which must be robust to attacks, and especially so to those attacks that can be deployed in the real world. On the one hand, our study demonstrates that Adversarial Scratches are effective in a variety of scenarios, including attacks against a publicly available API, even though it requires modification of very few pixels. On the other hand, we have presented filtering-based countermeasures to mitigate the vulnerabilities originating from Adversarial Scratches, and have quantitatively assessed the impact of these defenses on the model's performance.

In this work, we have limited our scope to one pixel wide Bézier curves targeting image classifiers. Future works may on the one hand study the effects of such attacks against models addressing higher visual recognition tasks, such as object detection and segmentation. On the other hand, future works may expand on the concept of Adversarial Scratches, proposing deployable attacks based on different parametric models. We also plan on developing countermeasures to extended versions of Adversarial Scratches, as the proposed filtering techniques may not be effective. Another important direction is that of studying the robustness of Adversarial Scratches to realistic changes in the image acquisition, including pose, light conditions, and background contents. In this work, we focused on applying scratches to single views of traffic signs, however, attacks that are successful regardless of acquisition settings may pose very serious threats to critical systems in the real-world, such as autonomous vehicles and AI-powered security cameras. Also in this case, we plan to develop new countermeasures, as such robust attacks may not be affected by simple filtering.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The link to the code and data is available in the manuscript.

References

[1] B. A. Barsky, Arbitrary subdivision of Bézier curves, 1985.

[2] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, F. Roli, Evasion attacks against machine learning at test time, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 2013, pp. 387–402.

[3] B. Biggio, B. Nelson, P. Laskov, Poisoning attacks against support vector machines, arXiv:1206.6389, 2012.

[4] W. Brendel, J. Rauber, M. Bethge, Decision-based adversarial attacks: reliable attacks against black-box machine learning models, arXiv:1712.04248, 2017.

[5] A.E. Cinà, A. Torcinovich, M. Pelillo, A black-box adversarial attack for poisoning clustering, Pattern Recognit. 122 (2022) 108306.

[6] F. Croce, M. Andriushchenko, N.D. Singh, N. Flammarion, M. Hein, Sparse-RS: a versatile framework for query-efficient sparse black-box adversarial attacks, in: Conference on Artificial Intelligence, vol. 36, 2022, pp. 6437–6445.

[7] F. Croce, M. Hein, Sparse and imperceivable adversarial attacks, in: International Conference on Computer Vision, 2019, pp. 4724–4732.

[8] T. Dai, Y. Feng, B. Chen, J. Lu, S.-T. Xia, Deep image prior based defense against adversarial examples, Pattern Recognit. 122 (2022) 108249.

[9] I. Evtimov, W. Cui, E. Kamar, E. Kiciman, T. Kohno, J. Li, Security and machine learning in the real world, arXiv:2007.07205, 2020.

[10] A. Ghosh, S.S. Mullick, S. Datta, S. Das, A.K. Das, R. Mallipeddi, A black-box adversarial attack strategy with adjustable sparsity and generalizability for deep image classifiers, Pattern Recognit. 122 (2022) 108279.

[11] C. Guo, J. Gardner, Y. You, A.G. Wilson, K. Weinberger, Simple black-box adversarial attacks, in: International Conference on Machine Learning, 2019, pp. 2484–2493.

[12] K. Han, B. Xia, Y. Li, 2: adversarial domain adaptation to defense with adversarial perturbation removal, Pattern Recognit. 122 (2022) 108303.

[13] M. Hazewinkel, Encyclopaedia of Mathematics: Supplement 3, Springer, 2011.

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

[15] Y. LeCun, P. Haffner, L. Bottou, Y. Bengio, Object recognition with gradient-based learning, in: Shape, Contour and Grouping in Computer Vision, Springer, 1999, pp. 319–345.

[16] D. Li, J. Zhang, K. Huang, Universal adversarial perturbations against object detection, Pattern Recognit. 110 (2021) 107584.

[17] H. Li, H. Li, H. Zhang, W. Yuan, Black-box attack against handwritten signature verification with region-restricted adversarial perturbations, Pattern Recognit. 111 (2021) 107689.

[18] Y. Li, B. Wu, Y. Feng, Y. Fan, Y. Jiang, Z. Li, S.-T. Xia, Semi-supervised robust training with generalized perturbed neighborhood, Pattern Recognit. 124 (2022) 108472.

[19] Microsoft, Cognitive services image captioning API, 2020, https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision.

[20] A. Modas, S.-M. Moosavi-Dezfooli, P. Frossard, Sparsefool: a few pixels make a big difference, in: Conference on Computer Vision and Pattern Recognition, 2019, pp. 9087–9096.

[21] S. Moon, G. An, H.O. Song, Parsimonious black-box adversarial attacks via efficient combinatorial optimization, in: International Conference on Machine Learning, 2019, pp. 4636–4645.

[22] N. Narodytska, S.P. Kasiviswanathan, Simple black-box adversarial attacks on deep neural networks, in: Conference on Computer Vision and Pattern Recognition Workshops, vol. 2, 2017, p. 2.

[23] National Nature Science Foundation of China, Traffic sign recognition database, 2020, http://www.nlpr.ia.ac.cn/pal/trafficdata/recognition.html.

[24] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z.B. Celik, A. Swami, The limitations of deep learning in adversarial settings, in: European Symposium on Security and Privacy, 2016, pp. 372–387.

[25] K.V. Price, Differential Evolution, Springer Berlin Heidelberg, 2013, pp. 187–214.

[26] S. Rao, D. Stutz, B. Schiele, Adversarial training against location-optimized adversarial patches, in: European Conference on Computer Vision, 2020, pp. 429–448.

[27] J. Rapin, O. Teytaud, Nevergrad - A gradient-free optimization platform, 2018, https://GitHub.com/FacebookResearch/Nevergrad.

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252.

[29] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, arXiv:1312.6199, 2013.

[30] Y. Xiao, C.-M. Pun, B. Liu, Fooling deep neural detection networks with adaptive object-oriented adversarial perturbation, Pattern Recognit. 115 (2021) 107903.

[31] 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, 2020, pp. 681–698.

[32] Z.B. Zabinsky, et al., Random Search Algorithms, Citeseer, 2009.

[33] M. Zambrano-Bigiarini, M. Clerc, R. Rojas, Standard particle swarm optimisation 2011 at CEC-2013: a baseline for future PSO improvements, in: Congress on Evolutionary Computation, 2013, pp. 2337–2344.

Loris Giulivi is a Ph.D. student at Politecnico di Milano. His research interests lie in deep learning and computer vision, with a particular focus on Explainable Artificial Intelligence and safe applications of deep learning in industrial contexts.

Malhar Jere is a graduate student at UC San Diego, where he works in ACESLab. His research interest lie in understanding deep neural networks to make them more trustworthy, safe, secure and private.

Loris Rossi worked on adversarial machine learning during his visiting research period at UC San Diego. He is now working on deep learning models as a Machine Learning Engineer at Amazon, after completing the master's degree at Politecnico di Milano.

Farinaz Koushanfar (Fellow, IEEE) is the Henry Booker Scholar Professor of electrical and computer engineering at UC San Diego, where she is the founding co-director of the MICS center. Her research interests include secure and robust ML for intelligent systems, automated co-design and optimization of full-stack solutions, and privacy-preserving computing.

Gabriela Ciocarlie, Ph.D., is an associate professor at The University of Texas at San Antonio and Vice President for Securing Automation and Secure Manufacturing Architecture for CyManII. Her expertise is in anomaly detection, distributed alert correlation, network and application level security, cyber physical systems security and distributed system security.

Briland Hitaj is a Computer Scientist at SRI International. His expertise is in security and privacy in deep learning systems. Prior to joining SRI International, he was a Visiting Research Scholar at Stevens Institute of Technology. Dr. Hitaj obtained his Ph.D. in Computer Science from Sapienza University of Rome, Italy.

Giacomo Boracchi is an Associate Professor at Politecnico di Milano, where he also received the Ph.D. in Information Technology (2008). His research interests concern image processing and machine learning, and in particular algorithms for image restoration and analysis, change/anomaly detection, domain adaptation.