Adversarial Scratches: Deployable Attacks to CNN Classifiers

Loris Giuliviab,∗, Malhar Jereb, Loris Rossia, Farinaz Koushanfarc, Gabriela Ciocarlied, Briland Hitajd, Giacomo Boracchiab

aPolitecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milano, Italy
bUniversity of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0021, USA
cUniversity of Texas San Antonio, One UTSA Circle, San Antonio, TX 78249, USA
dSRI 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 L0 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.

Keywords: Adversarial Perturbations, Adversarial Attacks, Deep Learning, Convolutional Neural Networks, Bézier Curves

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 [20], 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 [6] and [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 de-
Deployable and non-deployable perturbations

| Perturbation | Description |
|--------------|-------------|
| $L_1$, $L_2$, $L_\infty$ | $L_1$, $L_2$, and $L_\infty$ attacks are not deployable as they potentially modify all the image's pixels. |
| Frame | Frame attacks are not deployable as they modify pixels outside of the target object. |
| $L_0$ | Sparse attacks, even if localized to the target, are not deployable as they modify several regions in the image. It is unfeasible to apply this perturbation. |
| Patch | The Patch and Scratch attacks are deployable. The perturbations affects spatially contiguous regions, which are entirely contained in the target. |
| Scratch | |

Figure 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.
ployable. 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, $L_1, L_2, L_\infty$ and Frame attacks in Figure 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 Figure 1). Attacks such as [6, 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 figure 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 Figure 1 and in Figure 2. Adversarial Scratches are constituted of parametric curves, resembling graffiti or small damages when applied to 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 deployable attacks on in-the-wild traffic signs.

Our code and datasets are made publicly available here.

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 Microsoft’s 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 \( \mathbf{x} \in [0, 1]^{w \times h \times c} \) 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 \( C(\mathbf{x}) = \arg \max_i f(\mathbf{x})_i \). 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 \( \mathbf{x}' \in [0, 1]^{w \times h \times c} \) which, according to a distance metric \( L \) and a distance threshold \( \delta \), is close to the original sample \( \mathbf{x} \):

\[
L(\mathbf{x}', \mathbf{x}) < \delta,
\]

such that the model is fooled, which means:

\[
C(\mathbf{x}') \neq y.
\]

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) (Figure 1). Another requirement for deploya- bility 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. Figure 3 shows how this region is used.
2.2. Prior Works and Categorization of Attacks

Biggio et al. [3][2] 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 [3]. 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 \( x' \), and ii) the search strategy used to explore this space.

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 \( x' \) and observe its output. Black-box attacks can also be decomposed into score-based attacks, when the full class score vector \( f(x') \) is provided [11], or decision-based ones [4], when only the predicted label \( C(x') \) 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 \( ||x' - x||_p \). \( L_0 \) 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 \( L_\infty \) attacks the largest pixel-wise difference between the two. An adversarial sample \( x' \) 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 \( x \) (1).

Recent literature has proposed attacks in the \( L_1 \), \( L_2 \), and \( L_\infty \) norm constraint scenarios. Perhaps the most common are \( L_\infty \) attacks, such as [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 \( L_\infty \) 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 (Figure 1). \( L_1, L_2, L_\infty \) attacks, resulting in non-deployable perturbations. In contrast, attacks of \( L_0 \) nature such as [22, 6] 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 \( C(x') = y' \), and untargeted ones, where the objective is simply to induce misclassification, independently of the resulting class, such that \( C(x') \neq y = C(x) \). For this categorization, we cite Sparse-RS [6], as it displays both targeted and un-
targeted 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 adversarial sample $x'$ belonging to the ground truth class $y$, and the maximum posterior probability $\max_{i \neq y}(f(x')_i)$ of $x'$ belonging to a class other than $y$. The sample $x'$ is misclassified when $\max_{i \neq y}(f(x')_i) > f(x')_y$.

Therefore, $x'$ is an adversarial sample for $x$ when:

$$L_f(x, x') < 0.$$ (6)

We define the margin loss $L_f$ as in [6] as the difference between the posterior probability $f(x')_y$ of $x'$ belonging to the ground truth class $y$, and the maximum posterior probability $\max_{i \neq y}(f(x')_i)$ of $x'$ belonging to a class other than $y$. The sample $x'$ is adversarial for $x$ when:

$$L_f(x, x') < 0.$$ (6)

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

3. Methodology

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

$$L_f(x, x') = f(x')_y - \max_{i \neq y}(f(x')_i)$$ (3)

subject to the $L_0$ bound:

$$\|x' - x\|_0 \leq k,$$ (4)

and to the localization constraint:

$$r[i, j] = 0 \implies (x'[i, j] = x[i, j]).$$ (5)

We define the margin loss $L_f$ as in [6] as the difference between the posterior probability $f(x')_y$ of $x'$ belonging to the ground truth class $y$, and the maximum posterior probability $\max_{i \neq y}(f(x')_i)$ of $x'$ belonging to a class other than $y$. The sample $x'$ is adversarial for $x$ when:

$$L_f(x, x') < 0.$$ (6)

We solve the constrained optimization problem through an iterative optimization procedure. The result is an image $x'$ obtained by superimposing a scratch $B$ to the original sample $x$, where $B$ is modeled as a Bézier curve identified by parameter vector $v$. Figure 3 visualizes how the perturbation is computed starting from this parameter configuration, highlighting the usage of region $r$. 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 (Figure 4) is defined as:

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

As shown in Figure 4, second-order Bézier curves are solely defined by their control points $P_0, P_1, P_2$.

\footnote{This holds also when $x$ is not correctly classified ($C(x) \neq y$), and in this case we have $L_f(x, x) < 0$. However, $x$ 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 $x$.}

6
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, 2016 [22]    | local random search              | sparse                 | ✓                        | ✓                                    | ×                  |
| JSMA [24]                                   | substitute model                 | sparse                 | ✓                        | ✓                                    | ×                  |
| Boundary attack [4]                         | gaussian perturbation            | unstructured           | ✓                        | ×                                    | ×                  |
| Ilyas et al. 2018 [14]                      | natural evolution strategy       | unstructured           | ✓                        | ×                                    | ✓                  |
| CornerSearch [7]                            | coordinate-wise gradient estimation | sparse               | ×                        | ✓                                    | ×                  |
| SimBA [11]                                  | Fourier coefficients             | unstructured           | ✓                        | ✓                                    | ×                  |
| Parsimonious Black-Box Adversarial Attacks [27] | 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                | ✓                        | ✓                                    | ✓                  |

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].

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 $v \in \mathbb{R}^3$ (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.
Figure 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.

Figure 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 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 Figure 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 $v \in \mathbb{R}^9$. 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 $L_f$ 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) [23], and neuro-genetic optimization (NGO) [27]. RS and DE were chosen as they were used in related works [9, 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 \( x \), the query threshold \( f(x) \), the model’s prediction for the original sample \( y^* \), the target object. For clarity, we only describe the application of a single scratch. Section 4.2 details the extension of this procedure to multiple scratches.

Algorithm 1 general attack procedure

**Require:** \( x, y, MAX\_ITER, f, optimizer, k, r \)

1. \( iter \leftarrow 0 \)
2. \( C \leftarrow \arg \max f(x) \) \( \triangleright \) Predict class for original image
3. for \( i = 1 \) to \( MAX\_ITER \) do
   4. \( v \leftarrow optimizer.ask() \) \( \triangleright \) Obtain a solution from the optimizer
   5. \( v^* \leftarrow clip(v, k, r) \) \( \triangleright \) Clip the scratch (Algorithm 2)
   6. \( B^* \leftarrow Bézier(v^*, ..., v^*_8) \) \( \triangleright \) Sample Bézier
   7. \( c \leftarrow (v^*_8, v^*_7, v^*_6) \) \( \triangleright \) Apply scratch to obtain adversarial image \( x^{v^*} \)
   8. \( x^{v^*} \leftarrow x \)
   9. for \( j = 1 \) to \( |B^*| \) do
      10. \( x^{v^*}[b^*_j] \leftarrow c \) \( \triangleright \) Set pixels on the scratch support to the scratch color
   11. end for
   12. \( l \leftarrow f_y(x^{v^*}) - \max f_x(x^{v^*}) \) \( \triangleright \) Feed \( x^{v^*} \) to the model and compute loss
   13. if \( l < 0 \) then
      14. \( iter \leftarrow i \)
      15. break
   16. end if
   17. \( optimizer.tell(v, l) \) \( \triangleright \) 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

sampling curve (7) and ii) its color \( c = \{v^*_8, v^*_7, v^*_6\} \) (Lines 6-7). The tentative adversarial image \( x^{v^*} \) is then obtained by changing the color of \( x \) for each pixel location in the scratch support \( B^* \) to the scratch color \( c \) (Lines 8-11):

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

Thus, we compute the margin loss (3) by feeding the tentative adversarial image \( x^{v^*} \) to the target model \( f \) (Line 12), and if the attack is successful (9),
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 10000. 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 \( L_f(x, x^v) \) where the argument is the sample with the clipped perturbation \( x^v \) instead of \( x \). This choice is justified since clipping does not modify feasible solutions, therefore, for all feasible \( x^v \),

\[
L_f(x, x^v) = L_f(x, x^v^*),
\]

### 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 \( r \), and returns a new parameter configuration \( v^\ast \). 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 \( B^\ast = \{b_p, b_{p+1}, \ldots, b_{p+q}\} \) of \( B \) which is entirely contained in the region \( r \), starting at the first pixel of \( B \) belonging to the mask. The set of all Bézier curves is closed with respect to arbitrary subdivision [7], thus, given any two points \( P_0^*, P_2^* \) along a Bézier

```latex
\textbf{Algorithm 2} Scratch clipping

\textbf{Require:} \( v, k, r \)
1: \( P_0 = \{v_0, v_1\}, P_1 = \{v_2, v_3\}, P_2 = \{v_4, v_5\} \) \triangleright Input Bézier control points
2: \( \text{flag} \leftarrow 0, d \leftarrow 0, B^\ast \leftarrow \emptyset \) \triangleright Initialize flags and scratch support
3: \( B, T \leftarrow \text{Bézier}(v_0, \ldots, v_5) \)
4: \( \text{for } j = 1 \text{ to } |B| \text{ do} \)
5: \( \quad \text{if} \text{ flag} == 0 \text{ and } r(B) == 1 \text{ then} \triangleright \text{Beginning of the segment found} \)
6: \( \quad \quad \text{if} \text{ flag} == 1 \text{ then} \)
7: \( \quad \quad \quad \text{break} \)
8: \( \quad \text{else} \)
9: \( \quad \quad d++ \)
10: \( \quad \quad B^\ast \leftarrow B^\ast \cup b_j \triangleright \text{Add the location to the support} \)
11: \( \text{end if} \)
12: \( \text{end if} \)
13: \( \text{end for} \)
14: \( P_0^* \leftarrow (b_1, b_1^*, b_2) \)
15: \( P_1^* \leftarrow (1 - t_p)P_1 + t_pP_2 \)
16: \( P_1^\prime \leftarrow (1 - t_{p+q})b_0 + t_{p+q}P_1^* \)
17: \( P_2^* \leftarrow (b_{B^\ast}, b_{B^\ast}, b_{B^\ast,y}) \)
18: \( v \leftarrow \{P_0^* x, P_0^* y, P_1^*, P_1^\prime x, P_1^\prime y, P_2^* x, P_2^* y, v_6, v_7, v_8\} \)
19: \( \text{return } v^\ast \)
```

The ordering is given by the parametric location \( t \) along the Bézier curve [7].
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 Equation 8 for the case with \( n > 1 \) scratches, so that the image where scratches \( 1 \ldots s \) are applied is defined as follows:

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

where \( x^{(0)} = x \) and the end result is \( x^{(n)} = x^{(s)} \).

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). Lastly, we 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.

**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.

**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 10000. The metrics are defined as follows:
- **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.

- **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.

- **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.

**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 $L_\infty$ attacks, since given a specific $L_0$ bound it is impossible to define $L_1$, $L_2$, and $L_\infty$ 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 $L_\infty$ attacks cannot be deployed.

**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.

**Considered Methods:** We compare against Patch-RS [6], PatchAttack [31], and LOAP w/ GE [26, 6], 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 (Figure 5, left), differing only 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 [6], resulting in the black-box LOAP w/ GE.

**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 $20 \times 20$ 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 $L_0 = 399$ (Figure 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.

**Experiments on TSRD:** To better tests the deployability of the attacks, we design an experiment on the TSRD [22] 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 fine-tuned to 98% test accuracy on the TSRD dataset.

**Considered Methods:** For this experiment, we
The attack perturbs a square region of size $\sqrt{k}$ pixels. The attack places three scratches, one pixel wide and $k/3$ pixels long. The attack modifies $k$ pixels anywhere on the image.

Figure 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 [4].

Table 3: Comparison to deployable attacks. Results for Patch-RS, LOAP w/ GE and PatchAttack are reported from [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 | 10000 ±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 |

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 modifications to Sparse-RS’ code. Figure 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

...
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 \((L_0 = 399)\) |

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.

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 \(x\), provides a caption describing the image’s content and the model’s confidence \(h(x)\). Since the margin loss (3) cannot be computed in the captioning scenario, we solve the optimization problem by minimizing the loss:

\[
\mathcal{L}_g = h(x). \tag{10}
\]

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 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 Figure 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 generated scratches are shown in Figure 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 compare to are more general than Adversarial Scratches and can exploit a much wider attack surface.

Considered Methods: We compare to the “any-pixel” attack from Sparse-RS (Figure 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.

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.

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 differ-
Figure 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.

ent 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.
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% | 392 ±38 | 27 ±3 | Three Bézier, each 133px long            |
| “any-pixel”       | 99.9% ±0.2% | 154 ±6 | 25 ±1 | Any 400 pixels in the image              |
| $L_0 = 50$        |       |       |       |                                         |
| 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             |

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 1 000 samples of the ImageNet dataset.

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.

Scratch quantity: The number of scratches is a parameter that significantly influences deployability. As displayed in Figure 8, 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.

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.

Order of Bézier: We test Adversarial Scratches modeled as Bézier curves of varying order, as displayed in Figure 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 (11) can be generalized to express a Bézier of order $n$ as follows:

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

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
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]          |

![Original 1 scratch 2 scratches 3 scratches 4 scratches 5 scratches](image)

Figure 8: $L_0 = 400$ attacks with different number of scratches

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 ± 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  |

![Original 1st order 2nd order 3rd order 4th order](image)

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

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

**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;
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   |

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  |

- 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.

Figure 10 shows example scratches 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.

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],
Table 10 shows that both JPEG compression and median filtering are effective defenses, with median filtering providing the highest recovery rate (77.1%). Figure 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 \( x' \) that are classified as belonging to a specific target class \( y' \), namely \( C(x') = y' \). To obtain targeted adversarial scratches, we replace the optimization objective (3) with a cross entropy loss \( \mathcal{H} \), where the target vector \( 1_{y'} \) is all zeroes except for a one in position \( y' \):

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

In the untargeted scenario, the lower bound \( \mathcal{L}_f(x, x') \) identifies the threshold below which an image is adversarial. This is not the case for targeted attacks 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:

- **Median filtering** with varying quality factors.
- **JPEG compression** separately to each channel with a kernel size of \( 3 \times 3 \) 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:

\[
RR = \frac{|\{C(x) = y\} \cap \{C(x') \neq y\} \cap \{C(d(x')) = y\}|}{|\{C(x) = y\} \cap \{C(x') \neq y\}|},
\]

(12)

where \( X' = \{x_1', \ldots, x_n'\} \) is the set of perturbed images (possibly including those for which the attack was not successful) for samples \( X = \{x_1, ..., x_n\} \).

**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).

\[\text{If the filtered image is classified differently than the adversarial one } C(x') \neq C(d(x)), \text{ but the resulting class is not the target } C(d(x)) \neq y, \text{ the image is not considered to be recovered.}\]
Table 10: Recovery Rate and effects of the defenses on model performance.

| Defense strategy                      | Recovery Rate | Original images | Model performance | Performance delta |
|---------------------------------------|---------------|-----------------|-------------------|-------------------|
| No defense (baseline)                 | 0%            | 76.7%           | 76.7%             | 0%                |
| Median filtering, 3 × 3 kernel        | 77.1%         | 73.0%           | 73.0%             | -3.7%             |
| JPEG, quality = 85                    | 41.4%         | 74.4%           | 74.4%             | -2.3%             |
| JPEG, quality = 90                    | 39.6%         | 74.9%           | 74.9%             | -1.8%             |
| JPEG, quality = 95                    | 32.6%         | 75.7%           | 75.7%             | -1.0%             |
| JPEG, quality = 99                    | 25.7%         | 76.1%           | 76.1%             | -0.6%             |

attacks, where classification (and thus attack success) must be explicitly verified by checking whether $C(x') = y'$.

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.

8. Acknowledgements

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. We gratefully acknowledge the support of NVIDIA for the four A6000 GPUs granted through the Applied Research Accelerator Program to Politecnico di Milano.

References

[1] Brian A Barsky. 1985. *Arbitrary subdivision of Bézier curves.*

[2] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*. 387–402.
[3] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. *arXiv preprint* 1206.6389.

[4] Wieland Brendel, Jonas Rauber, and Matthias Bethge. 2017. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint* 1712.04248.

[5] Antonio Emanuele Cinà, Alessandro Torcianovich, and Marcello Pelillo. 2022. A black-box adversarial attack for poisoning clustering. *Pattern Recognition* 122, 108306.

[6] Francesco Croce, Maksym Andriushchenko, Naman D Singh, Nicolas Flammarion, and Matthias Hein. 2022. Sparse-rs: a versatile framework for query-efficient sparse black-box adversarial attacks. In *Conference on Artificial Intelligence*, Vol. 36. 6437–6445.

[7] Francesco Croce and Matthias Hein. 2019. Sparse and imperceivable adversarial attacks. In *International Conference on Computer Vision*. 4724–4732.

[8] Tao Dai, Yan Feng, Bin Chen, Jian Lu, and Shu-Tao Xia. 2022. Deep image prior based defense against adversarial examples. *Pattern Recognition* 122, 108249.

[9] Ivan Evtimov, Weidong Cui, Ece Kamar, Emre Kiciman, Tadayoshi Kohno, and Jerry Li. 2020. Security and machine learning in the real world. *arXiv preprint* 2007.07205.

[10] Arka Ghosh, Sankha Subhra Mullick, Shoumak Datta, Swagatam Das, Asit Kr Das, and Ram Mohan Mallipeddi. 2022. A black-box adversarial attack strategy with adjustable sparsity and generalizability for deep image classifiers. *Pattern Recognition* 122, 108279.

[11] Chuan Guo, Jacob Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Weinberger. 2019. Simple black-box adversarial attacks. In *International Conference on Machine Learning*. 2484–2493.

[12] Keji Han, Bin Xia, and Yun Li. 2022. 2: Adversarial domain adaptation to defense with adversarial perturbation removal. *Pattern Recognition* 122, 108303.

[13] Michiel Hazewinkel. 2011. *Encyclopaedia of Mathematics: Supplement 3*. Springer.

[14] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. 2018. Black-box adversarial attacks with limited queries and information. In *International Conference on Machine Learning*. 2137–2146.

[15] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*. Springer, 319–345.

[16] Debang Li, Junge Zhang, and Kaiqi Huang. 2021. Universal adversarial perturbations against object detection. *Pattern Recognition* 110, 107584.

[17] Haoyang Li, Heng Li, Hansong Zhang, and Wei Yuan. 2021. Black-box attack against handwritten signature verification with region-restricted adversarial perturbations. *Pattern Recognition* 111, 107689.

[18] Yiming Li, Baoyuan Wu, Yan Feng, Yanbo Fan, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. 2022. Semi-supervised robust training with generalized perturbed neighborhood. *Pattern Recognition* 124, 108472.

[19] Microsoft. 2020. Cognitive Services Image Captioning API. https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision.

[20] Apostolos Modas, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. 2019. Sparsefool: a few pixels make a big difference. In *Conference on Computer Vision and Pattern Recognition*. 9087–9096.

[21] Seungyong Moon, Gaon An, and Hyun Oh Song. 2019. Parsimonious black-box adversarial attacks via efficient combinatorial optimization. In
International Conference on Machine Learning. 4636–4645.

[22] Nina Narodytska and Shiva Prasad Kavviswanathan. 2017. Simple Black-Box Adversarial Attacks on Deep Neural Networks.. In Conference on Computer Vision and Pattern Recognition Workshops, Vol. 2. 2.

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

[24] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In European symposium on security and privacy. 372–387.

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

[26] Sukrut Rao, David Stutz, and Bernt Schiele. 2020. Adversarial training against location-optimized adversarial patches. In European Conference on Computer Vision. 429–448.

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

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhenghe Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. International Journal of Computer Vision 115, 3, 211–252.

[29] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. arXiv preprint 1312.6199.

[30] Yatie Xiao, Chi-Man Pun, and Bo Liu. 2021. Fooling deep neural detection networks with adaptive object-oriented adversarial perturbation. Pattern Recognition 115, 107903.

[31] Chenglin Yang, Adam Kortylewski, Cihang Xie, Yinzhi Cao, and Alan Yuille. 2020. Patchattack: A black-box texture-based attack with reinforcement learning. In European Conference on Computer Vision. 681–698.

[32] Zelda B Zabinsky et al. 2009. Random search algorithms. Department of Industrial and Systems Engineering, University of Washington, USA.

[33] Mauricio Zambrano-Bigiarini, Maurice Clerc, and Rodrigo Rojas. 2013. Standard Particle Swarm Optimisation 2011 at CEC-2013: A baseline for future PSO improvements. Congress on Evolutionary Computation, 2337–2344.