Appendix: RFLA: A Stealthy Reflected Light Adversarial Attack in the Physical World

Donghua Wang[1], Wen Yao*[2], Tingsong Jiang*[2], Chao Li[3], and Xiaoqian Chen[2]

[1]College of Computer Science and Technology, Zhejiang University
[2]Defense Innovation Institute, Chinese Academy of Military Science
[3]School of Artificial Intelligence, Xidian University
wangdonghua@zju.edu.cn, {wendy0782,lichaoedu}@126.com, tingsong@pku.edu.cn,
chenxiaoqian@nudt.edu.cn

A. Implementation details

In this section, we first introduce the lower and upper bounds of the particle and velocity. Recall that a particle q represents an optimization variable tuple $(x, y, r, \alpha, red, green, blue, a_1)$, where the lower bound is (0, 0, 10, 0, 0, 0, 0, 0, 0, 0) and the upper bound is (H/2, W/2, 0.4×min(W, H), 0.7, 255, 255, 255, 360) except for the line. As for the RFLA-Line, the transparency α is set to 1. The reason is that the modification of the clean image caused by the RFLA-Line geometry shape merely has a few numbers of pixels, even if the original pixels are replaced. Velocity controls the movement speed of particles: a large velocity speed would early lead the particle to reach the bound, while a small velocity makes the particle move slowly, requiring more optimizing time. Thus, we set them in terms of the concrete meaning of the variable. Specifically, we set the upper bound of velocity as follows: coordination and radius of the circle are set to 5, 5, and 10; the transparency α is set to 0.05; the color is set to 5; the angle is set to 10, i.e., the initialization of velocity's upper bound is set to $v_{upper} = (5, 5, 10, 0.05, 5, 5, 5, 10)$. In contrast, the lower bound is set to $v_{lower} = -v_{upper}$.

The initialization of the proposed method is significant to optimization, which describes the variable that requires to be optimized. Algorithm A1 describes the initialization of particles and the corresponding velocity generation for different geometries. Specifically, we generate population size S particles, i.e., the center $o(x, y)$ and radius r of the circle. For each specific circle, we generate subpopulation size S_{sub} geometries, which consists of $\alpha, red, green, blue, a_1$. Note that we fixed the coordinate and radius when spawning different topologies of the same geometrical shape at a specific circle. Therefore, we devise a novel variable to record which circle can generate the optimal geometrical shapes from an overall viewpoint, i.e., the q_{sgbest}, which is defined as

$$q_{sgbest} = \arg\min_{i \in S} \sum_{j=1}^{S_{sub}} F(q_{i,j}). \tag{1}$$

Moreover, the defination of the q_{gbest} and q_{pbest} is expressed as follow

$$q_{gbest} = \arg\min_{i \in S, j \in S_{sub}} F(q_{i,j}). \tag{2}$$

$$q^i_{pbest} = \arg\min_{j \in S_{sub}} F(q_{i,j}). \tag{3}$$

After generating the particle, we use the Algorithm A2 to generate adversarial examples. Specifically, we first obtain the point on the circle by cosine and sine function with the coordinate and radius and an angle, then calculate its symmetry point with respect to the center points. Repets until generating enough points that the geometry shape required. Then, we sort the point set to avoid engendering the intercross edge. Finally, we use the OpenCV package to plot the geometry on the clean image.

B. Experiment and result analysis

Model attention analysis can reveal how the attack algorithm works. To provide a more complete visualization comparison results of different attack methods. We use the Grad-CAM tool to analyze changes in the class activation map caused by different attack algorithms. Specifically, we focus on the CAM of the original prediction class since the predicted class of the adversarial examples is different from

*Corresponding Author.

Algorithm A1 Population Initialization

Input: mask M, population size S, sub population size S_{sub}

Output: $q_{i,j}, v_{i,j}$, $i \in [1, ..., S], j \in [1, ..., S_{sub}]$

$q \leftarrow []$
$v \leftarrow []$
for $i = 1, ..., S$ **do**
  Randomly sample a radius r in $[r_{lower}, r_{upper}]$
  Sample a center (x, y) of the circle by $[r, x_{upper} - r]$ and $[r, y_{upper} - r]$
  while $M(x, y) == 0$ **do**
    Resample a center (x, y) of the circle
  end while
  Random initialize v_x, v_y, v_r from $[V_{lower}, V_{upper}]$
  for $j = 0, ..., S_{sub}$ **do**
    Initialize a transparency α, a angle a_1 and fill color $(red, green, blue)$
    Initialize velocity $v_\alpha, v_{a_1}, v_{red}, v_{green}, v_{blue}$
    if shape == line **then**
      $q_{i,j} \leftarrow (x, y, r, \alpha, red, green, blue, a_1)$
      $v_{i,j} \leftarrow (v_x, v_y, v_r, v_\alpha, v_{red}, v_{green}, v_{blue}, v_{a_1})$
    end if
    if shape == triangle **or** shape == rectangle **then**
      Initialize a new angle a_1
      Initialize a new angle velocity v_{a_2}
      $q_{i,j} \leftarrow (x, y, r, \alpha, red, green, blue a_1, a_2)$
      $v_{i,j} \leftarrow (v_x, v_y, v_r, v_\alpha, v_{red}, v_{green}, v_{blue}, v_{a_1}, v_{a_2})$
    end if
    if shape == pentagon **or** shape == hexagon **then**
      Initialize a new angle a_1 and a_2
      Initialize a new angle velocity v_{a_2}, v_{a_3}
      $q_{i,j} \leftarrow (x, y, r, \alpha, red, green, blue, a_1, a_2, a_3)$
      $v_{i,j} \leftarrow (v_x, v_y, v_r, v_\alpha, v_{red}, v_{green}, v_{blue}, v_{a_1}, v_{a_2}, v_{a_3})$
    end if
  end for
end for

Algorithm A2 Generate Adversarial Examples

Input: Populations q

Output: x_{adv}

$x_{adv} \leftarrow []$
for $i = 1, ..., S$ **do**
  for $j = 1, ..., S_{sub}$ **do**
    if shape == line **then**
      $(x, y, radius, \alpha, r, g, b, a_1) \leftarrow q_{i,j}$
      Get the point set p_1, p_1'
    end if
    if shape == triangle **then**
      $(x, y, radius, \alpha, r, g, b, a_1, a_2) \leftarrow q_{i,j}$
      Get the point set p_1, p_2, p_1'
    end if
    if shape == triangle **then**
      $(x, y, radius, \alpha, r, g, b, a_1, a_2) \leftarrow q_{i,j}$
      Get the point set p_1, p_2, p_1', p_2'
    end if
    if shape == pentagon **then**
      $(x, y, radius, \alpha, r, g, b, a_1, a_2, a_2) \leftarrow q_{i,j}$
      Get the point set $p_1, p_2, p_3, p_1', p_3'$
    end if
    if shape == hexagon **then**
      $(x, y, radius, \alpha, r, g, b, a_1, a_2, a_2) \leftarrow q_{i,j}$
      Get the point set $p_1, p_2, p_3, p_1', p_2', p_3'$
    end if
    Sorted the point set via the Minimum Points Distance algorithm
    $x_{adv} \leftarrow$ call function *fillpoly* of OpenCV-Package with the input x and particle q.
  end for
end for

the original prediction class. Thus, the CAM is also different. In contrast, the changes in the CAM of the original prediction can reflect the attack function. The comparison result is illustrated in Figure B2. As we can observe, the proposed method can disperse the CAM while the other method can not. Take a careful look at the CAM of the proposed method, the plotted geometry suppresses the original CAM, where the region is the region with semantic content. Therefore, the proposed method can obtain superior attack performance.

In addition, we provide the complete transferability comparison result in Table B1. As we can see, the proposed method achieves the best average ASR on both white-box and black-box attacks. One possible reason for the better transferability of the proposed method is that the proposed method can automatically locate the region of the model decision that is common for different models. Moreover, the proposed method is different from the full-pixel imperceptible perturbation, we generate adversarial examples by modifying partial image regions with the transparency color. Therefore, the image content of the modified region by the proposed method is maintained, which is the main difference compared to the patch-based attack (e.g., TPA and DAPatch).

Figure B1. Visualization comparison of adversarial examples generated by RFLA on ResNet50.

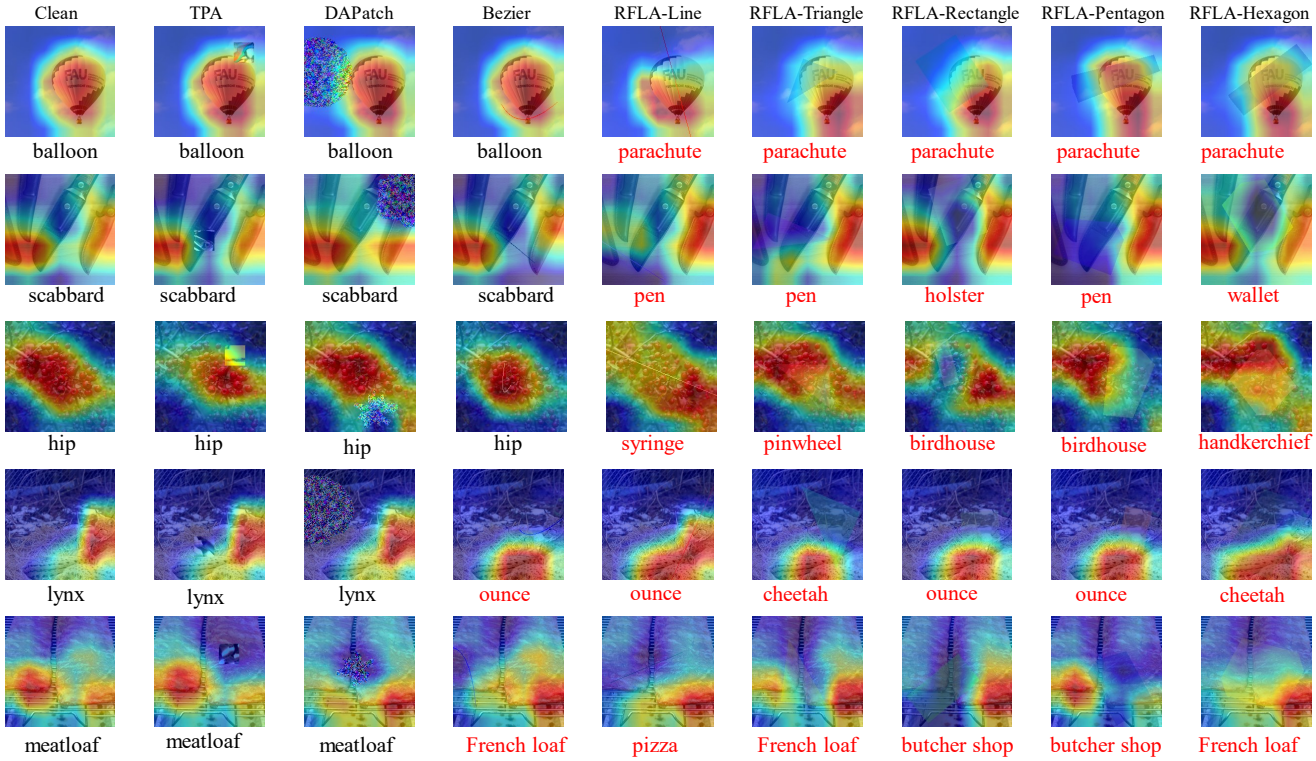

Figure B2. Model attention analysis of adversarial examples generated by different methods on ResNet50.

Table B1. Comparison results of transferability of adversarial examples generated by different methods in terms of ASR (%) on ImageNet-compatible dataset. *Item* indicates the white-box attack result, while the others are black-box results. The best results are highlighted with **bold**.

	method	RN50	VGG16	DN121	RNX50	WRN50	SN	AVG
RN50	TPA	*66.1*	**50.4**	42.7	**41.6**	38.6	55	45.66
	DAPatch	*74.3*	47.4	44.5	48	31.9	58.6	46.08
	Bezier	*72.4*	15.8	15.5	17.8	13.8	27.4	18.06
	RFLA-LINE	*76.9*	14.5	15.5	14.1	14.1	27.7	17.18
	RFLA-Triangle	*98.1*	32.2	33	29.4	30.6	48.5	34.74
	RFLA-Rectangle	*99.3*	43.8	44.8	37.5	40	60	45.22
	RFLA-Pengtagon	***99.6***	42.4	42.7	35.2	38.5	**62.8**	44.32
	RFLA-Hextagon	*99.5*	43.7	**46.9**	38.1	**44.2**	62.4	**47.06**
VGG16	TPA	30	*36*	29	26	22	46	30.6
	DAPatch	24.9	*71.6*	35	31.1	19.8	**76.4**	37.44
	Bezier	12	*77.7*	12.8	12.9	9.3	29.2	15.24
	RFLA-LINE	14.4	*77.4*	15.9	14.1	11.6	30	17.2
	RFLA-Triangle	28.7	*97.8*	26.9	23	23.2	47.8	29.92
	RFLA-Rectangle	35.1	*99.1*	30.1	28.1	30.4	52.3	35.2
	RFLA-Pengtagon	41.4	*99.1*	40.7	33.1	36.7	61.5	42.68
	RFLA-Hextagon	**43.2**	***99.5***	**43.1**	**37.1**	**39**	65.4	**45.56**
DN121	TPA	33.4	40.2	*40*	27.6	25.6	48.8	35.12
	DAPatch	28.8	**50.7**	*79.3*	37.2	21.1	74.4	42.44
	Bezier	16.6	14.4	*74.1*	14.1	10.6	27.5	16.64
	RFLA-LINE	15.4	14.9	*76.5*	15.7	11.7	26.9	16.92
	RFLA-Triangle	31.7	29.8	*97.2*	25.5	26.8	46.9	32.14
	RFLA-Rectangle	42.7	41.6	*99.2*	35	36.6	62.3	43.64
	RFLA-Pengtagon	45.2	43.1	*99.2*	34.8	38.5	60.3	44.38
	RFLA-Hextagon	**45.2**	43.8	***99.5***	**38**	**39.6**	**63.3**	**45.98**
RNX50	TPA	28.9	36	26.7	*25*	20.4	45.4	31.48
	DAPatch	32.3	**52.5**	42.3	*73.7*	26.4	28.8	36.46
	Bezier	18.1	16.8	16.4	*72.7*	12.4	26.6	18.06
	RFLA-LINE	17.6	15.5	17.5	*75.7*	14	27.1	18.34
	RFLA-Triangle	37.8	34.9	33.1	*97.2*	33.7	50	37.9
	RFLA-Rectangle	47.6	43.2	44.3	*98.9*	45.2	59.9	48.04
	RFLA-Pengtagon	47.5	45	46.4	*99.2*	45.2	61.9	49.2
	RFLA-Hextagon	**50.8**	44	**47.6**	***99.5***	**47**	**63.9**	**50.66**
WRN50	TPA	31.4	36.6	27.3	24.7	*23.4*	44	32.8
	DAPatch	40.2	**48.4**	**49.3**	**52.2**	*76.7*	60.8	50.18
	Bezier	19.4	17.7	14.6	18.7	*69.6*	25.1	19.1
	RFLA-LINE	17.8	15	16.8	16.2	*71.7*	25.2	18.2
	RFLA-Triangle	36.1	31.7	35.3	31.7	*98.1*	47.4	36.44
	RFLA-Rectangle	45.3	41.1	43.3	38.8	*99.1*	58.9	45.48
	RFLA-Pengtagon	48.9	41.8	46.8	40	*99.4*	61.4	47.78
	RFLA-Hextagon	**51**	47.3	48.1	40.6	*99.4*	**64.7**	**50.34**
SN	TPA	**29.8**	28.5	29.7	**23**	22.5	*44.6*	26.7
	DAPatch	17.3	**30.1**	23.8	22.2	14.3	*56*	21.54
	Bezier	10.3	12.2	11.3	9.1	8	*89.3*	10.18
	RFLA-LINE	10	11.5	12.5	8.8	8.5	*89.2*	10.26
	RFLA-Triangle	16.6	19.7	17.8	14.6	15.9	*99.5*	16.92
	RFLA-Rectangle	25.6	29.1	24.4	21.6	25	*99.8*	25.14
	RFLA-Pengtagon	25.7	28.8	28.2	**23**	24.3	*99.8*	26
	RFLA-Hextagon	26.9	29.3	**30.2**	21.8	**25.4**	*99.8*	**26.72**