Attacking JetBot

Souhaila Nouinou*, Alexis Motet*, Louis Morge-Rollet†, Frédéric Le Roy†
*ENSTA Bretagne, CSN 2023, F-29200 Brest, France
†ENSTA Bretagne, Lab-STICC, CNRS, UMR 6285, F-29200 Brest, France

***Abstract*—Used in medicine, autonomous driving and facial recognition, machine learning and deep learning algorithms are increasingly becoming part of our daily lives. Artificial intelligence experts and researchers need to push the limits of these algorithms by looking at what the flaws might be and whether they could be exploited by an attacker, to ensure that they are secure and usable by the general public.**

In a very concrete way, we attacked the neural network used by a robot to move autonomously with an attack that uses adversarial patches. Adversarial patches are small pieces of paper containing a pattern that confuses the image classifiers by making them indicate the wrong class. We managed to trap the robot's neural network in just under half of the static attacks, but this was enough to produce a few attacks on the moving robot that worked.

I. Introduction

Machine learning and deep learning are domains that have made great progress in recent years and are now capable of surpassing human capabilities in many areas such as facial recognition [1]. Behind these evolutions, the question of the trust that humans can give to these new tools still remains: for instance, the car manufacturer Tesla recently had to recall more than 360 000 vehicles because of flaws detected in the autonomous driving software [2].

In order to determine the level of confidence that can be given to these new machine learning and deep learning algorithms such as neural networks, it is necessary to study what the potential attacks are and what the defenses against such attacks might be.

One of the most prominent areas of deep learning is image classification. Image classification has been greatly developed thanks to a neural network architecture called AlexNet, which makes it possible to associate a label among a thousand other labels to an image with great precision [3]. Image classification is now used in concrete ways in fields such as medicine, automatic driving and surveillance.

Researchers have studied the potential weaknesses of a neural network classifying images and have proposed to create adversarial patches, which are small pieces of paper on which a pattern is printed that is supposed to disrupt the classification of the network when they are positioned in an image that one wants to attack.

The principle of this attack is that the addition of the patch in the field of view of the camera that feeds the neural network (also called a model) will deceive it by making it either indicate any other class than the ground label, or a particular targeted class that is not the ground label.

Access to the model and dataset on which it was trained makes the attack much easier, but black-box attacks can also work, training the patch from one model and another dataset but targeting the same class.

The goal of our project was to perform a physical attack using an adversarial patch on a model that a JetBot (an open-source robot based on NVIDIA Jetson Nano) uses to avoid obstacles [4]. The JetBot model comes from AlexNet and is based on transfer learning and training on a small dataset of about 500 images. The outputs of the model are "obstacle" or "no obstacle". The JetBot's obstacle avoidance algorithm is simple: each frame of the JetBot's camera enters the model and if the model labels the frame as an obstacle, then the JetBot turns left to avoid it. The frequency of the classifications is of the order of magnitude of ten milliseconds.

To perform our attack, we had access to the model and the training and test dataset (white-box attack), but we did not re-train the model or modify the dataset, nor did we change the model pre-processing (frame normalization). The attacked model and the dataset come from a work of two other students of ENSTA Bretagne realized during the year 2022.

One of the challenges of the project was to generate patches that could attack a model running with a camera with high distortion and high digital noise. The camera of the JetBot is also tilted downwards which causes a noticeable perspective effect on a patch placed straight in front of the robot.

II. Related Work

Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi and Justin Gilmer were the first to propose generating an adversarial patch that is scene-independent and robust to certain transformations like rotation and translation (2017) [5]. They used the Expectation Over Transformation (EOT) framework for the generation of such a patch [6].

Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer and Michael K. Reiter generated glasses to escape a classifier used for facial recognition (2016) [7]. They proposed to add a loss of total variation to make the glasses smoother and composed of colors printable by a common printer.

Simen Thys, Wiebe Van Ranst and Toon Goedemé sought to generate patches for an object detector (2019) [8]. They took the same total variation loss that Mahmood Sharif *et al.* used and slightly modified the printability loss. They also added other transformations such as noise and changes in brightness and contrast during patch generation to make it more robust to these perturbations.

Danny Karmon, Daniel Zoran, and Yoav Goldberg proposed modifying the objective function to generate a patch by targeting a given class and moving away from the class given by the model (2018) [9].

III. Contribution

A. Theoretical aspect of adversarial patch generation

Our problem is to find the patch that would work best in the physical world averaged over a set of physical transformations of the patch and determined scenes. The physical transformations can be for example a rotation of the patch in space or a change in the ambient lighting. The targeted scenes are scenes close to the training dataset of the model. We used for this the Expectation Over Transformation (EOT) framework which defines our objective function L that we want to minimize:

$$L(w) = \mathbb{E}_{t\sim T,\ x\sim X}[\textbf{softmaxloss}(f(t(x,w)),\hat{y})] \quad (1)$$

where f is the targeted model, $\hat{y}$ the targeted label, w a patch, x a sample of the distribution of images X, t a sample of the distribution of physical transformations T. A transformation t applied to an image x and a patch w gives an attacked image that we give to the model f [5], [7].

To find the patch that minimizes the objective in the set of potential patches, we use the stochastic gradient descent algorithm, which consists in approximating the gradient of the expectation with respect to patch w by the gradient of the *softmaxloss* with respect to patch w for a randomly taken pair (x,t). This gradient should in theory be obtained by back-propagation on the model.

However, a pair (x,t) and a given patch w should produce an attacked image $x_A = t(x,w)$, but we can only approximate the physical transformation t by a digital transformation $\tilde{t}$, and this last transformation gives us a digital attacked image $\tilde{x}_A = \tilde{t}(x,w)$. For example, the rotation of a patch in space will produce a perspective distortion of the patch on the image taken by the camera, and we reproduce (with a certain imprecision) this perspective distortion thanks to the knowledge of the intrinsic parameters of the camera obtained by the calibration. Thus, we will not be able to calculate exactly the gradient of the *softmaxloss* knowing (x,t) that we would have in reality but only an approximation of this gradient.

Also, the displacement transformations of the patch and the distortion are not differentiable, so we cannot obtain the gradient of the *softmaxloss* with respect to the patch w, but only the gradient of the *softmaxloss* with respect to the image under attack $\tilde{x}_A$. We then use a mask to recover in this last gradient the area corresponding to the patch to improve the patch included in the image during an iteration of the stochastic gradient descent. Finally, we recover the modified patch by applying the inverse of the non-differentiable transformations.

B. Practical aspect of the attack on the JetBot

The transformations that we applied to the patch are :

- Translation and rotation (yaw, pitch, roll) in space
- Radial distortion
- Contrast and brightness modification
- Adding Gaussian noise

Fig. 1. Comparison between a digital transformation (left) on a dataset image and a physical attack (right)

Fig. 1 shows a comparison between a digital transformation and a physical attack. In the rest of this section we will describe some practical aspects we have developed to generate our patches.

1) Tilt of the JetBot camera: The camera of the JetBot is tilted downwards, which causes a perspective effect on a patch that is shown facing the JetBot and perpendicular to the ground. This is why a transformation t necessarily includes a prior rotation of the patch to give it this perspective effect. We obtained the matrix of this rotation thanks to the calibration of the camera.

2) Radial Distortion: Distortion moves the values of a pixel from a position (x,y) to a position (x',y'). The radial distortion can be estimated as follows:

$$\begin{aligned} x' &= x(1 + k_1 r^2 + k_2 r^4) \\ y' &= y(1 + k_1 r^2 + k_2 r^4) \end{aligned} \quad (2)$$

where $r = \sqrt{x^2 + y^2}$ and $k1$, $k2$ are the distortion coefficients which can be obtained with camera calibration [10].

Because of the determined number of pixels in an image, x' and y' must be integers, which can cause several pixels of the source image to arrive at the same pixel of the digitally distorted image. In this case, we proposed to take the average of the values of the pixels that must overlap in the distorted image. The average is weighted by the distance between the theoretical position that a distorted pixel should have (floating x' and y') and the position that it actually has (integer x and y).

To recover the distorted patch after an iteration of the stochastic gradient descent in the case where the distorted patch pixel comes from several pixels of the source image, we take the previous weights and we modify each of the source pixels by taking the weighted average between the source value and the value coming from the distorted patch. This avoids creating color flats that would have occurred if we had placed the same value in each of the source pixels during recovery (we let the total variation manage the smoothness of the patch). If a pixel did not have an image due to the distortion because x or y went out of the frame, for example by being negative, then we keep the value of the source pixel in the image after undistortion. The Fig. 2 summarises our distortion algorithm and its inverse.

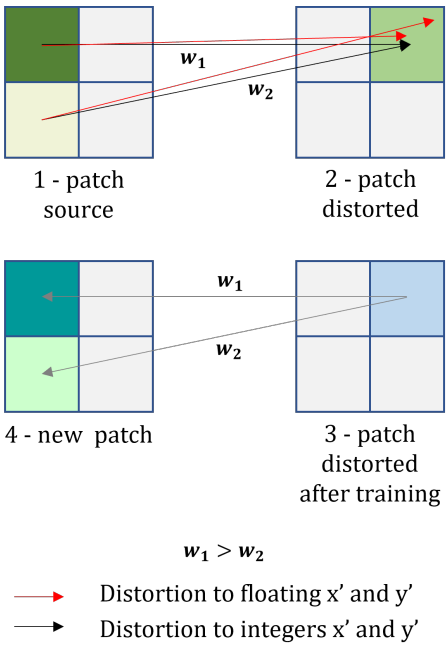

Fig. 2. Our distortion transformation and its inverse

Overall, our implementation of the distortion implies that compared to an injective transformation where each image pixel has only one antecedent pixel, one step of our algorithm does not go completely in the direction of the gradient descent (but the stochastic gradient is already an approximation of the gradient computed on the whole training dataset).

3) Loss: Since the output of our model is binary, we can improve the objective function by noting that we can modify the patch during the gradient descent both in the direction of the target class and away from the other class, instead of only going towards the target class. The *softmaxloss* for a pair (x, t) and a patch w is thus modified and becomes :

$$l(x,t,w) = \textbf{softmaxloss}(f(t(x,w)), \hat{y}) - \textbf{softmaxloss}(f(t(x,w)), \bar{y}) \quad (3)$$

where $\bar{y}$ is the non-targeted class.

4) Translation and rotation: Translations and rotations of the patch are done in space to make the effectiveness of the patch relatively independent of the position of the patch in the image, the relative size of the patch, and the viewing angle of the camera. These transformations are made thanks to the knowledge of the intrinsic parameters of the camera which allow us to pass from a 2D patch to a 3D patch and vice versa.

5) Gaussian noise and other transformations on the patch: The JetBot camera has significant digital noise that is visible to the naked eye. To make the patch robust to this noise, We added Gaussian noise to the patch at each iteration. The standard deviations of the noise were determined by taking a hundred photographs of a still image with the JetBot camera.

We also added a Gaussian blur and a variation in brightness and contrast to improve the robustness of the patch to variations in the outside world.

6) Attack zone: We have defined an attack area where the patch should work and therefore where translation of the patch is possible. This has the effect of improving the effectiveness of attacks (see end of sub-section D for explanation) without overly restricting the possible variety of attacks:

- The width restriction is not a great constraint on the diversity of attacks since the JetBot camera has a wide angle of view (the image distortion is the counterpart), which means that even an attack area with a small width already covers a wide angle of view
- The height restriction is not a real constraint either since we perform the attacks at a certain distance, we are only interested in the positions located at the top of the image because the bottom of the image is occupied by the ground

7) Total variation: To create a patch that would be smooth and without too many details that would not be captured by the camera, we modified our objective function by adding a total variation loss. It is given by :

$$L_{tv} = \sum_{i,j} \sqrt{(p_{i,j} - p_{i+1,j})^2 + (p_{i,j} - p_{i,j+1})^2} \quad (4)$$

where $p_{i,j}$ is the pixel of the patch at the row i and the column j [7], [8].

8) Printability: Since all the colors of an image are not necessarily printable by a common printer, we used a loss of printability that we added to the objective which makes it possible to quantify if the colors of the patch are close to a set of colors defined as printable. It is given by :

$$L_{print} = \sum_{p \in w} \min_{c_{print} \in C} |p - c_{print}| \quad (5)$$

where p is a pixel of the patch and c_{print} a color from our set of printable colors C [8].

C. Final objective function and proposed algorithm

Finally, the objective function that we want to optimize is :

$$L(w) = \mathbb{E}_{t \sim T,\ x \sim X}[l(x,t,w)] + \lambda_{tv} L_{tv} + \lambda_{print} L_{print} \quad (6)$$

where λ_{tv} and λ_{print} are hyperparameters that we have determined experimentally. In practice we have taken $\lambda_{tv} = 0.003$ and $\lambda_{print} = 0.002$.

D. Patch generation

We used Google Colab to train our patch, using a standard GPU. A training on for about ten epochs takes about 15 minutes, knowing that the dataset at our disposal contains about 500 images (training images and test images together). By using mini-batches of a few images at each step (mini-batch gradient descent), we accelerate the training without degrading the test results. It should be noted that the digital tests performed after each epoch are very theoretical, since our goal is to obtain a patch that works in the physical world and

Algorithm 1 Patch generation

for image, ground_label in dataset **do**
  if ground_label == target_class **then**
    continue
  end if
  pick a random transformation t
  transform the patch with t and get the mask m
  for $i <$ max_iterations **do**
    create the attacked image x_A from transformed patch and image
    compute the gradient of the *softmaxloss* with respect to the attacked image
    create the localized gradient at the patch using the mask m
    apply the localized gradient to x_A (gradient descent)
    clip to values of x_A to the image domain $[0, 1]$
  end for
  recover the untransformed patch in the attacked image by applying t^{-1}
  compute the gradients of the total variation and printability losses with respect to the patch
  apply both gradients to the patch (gradient descent)
end for

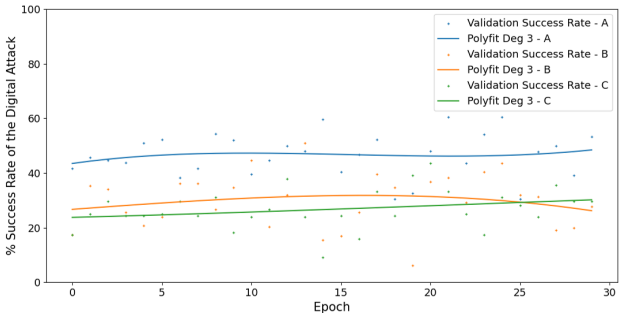

Fig. 3. Validation curves for different patch configurations

not only digitally. The real tests are those that are performed in practice with a printed patch, the digital tests overestimate the success of the patch in practice and are only an indication of how well the patch works in the physical world.

Fig. 3 contains the validation curves for different patch configurations: patch A is robust to 5 degree rotation and 0.9 scale reduction, patch B to 5 degree rotation and 0.7 scale reduction and patch C to 10 degree rotation and 0.7 scale reduction. In practice we used patch A for our tests.

We observe in the Fig. 3 that the attack success rate of a patch does not increase after a number of epochs. We think that this is because the patch quickly converges to a local or global maximum. It is important to understand that the perfect patch does not exist, but that we are only looking for the patch that works best on average given a certain distribution of transformations and a certain distribution of scenes. Overall, the more we ask of the patch, for example to be robust to yatch, pitch and roll rotations greater than 10 degrees, the more the best patch is likely to work less well, but the gradient descent will always indicate one of the best possible patches. In summary, there is a trade-off between the robustness of the patch to certain physical transformations and the effectiveness of the patch.

Fig. 4. Nine of the eleven scenes in the static test, attacked by patch A located at one of the three defined attack positions

E. Printing the patches

We printed the patches A, B and C on the thickest paper available to us (300 grams) and determined the side in centimeters of the patches to be printed by comparison with the images used by the calibration which contain white and black squares that are each 2.3 centimeters square. We had to do this because our training of the patch is done by specifying only a percentage of relative size that it must occupy on the screen and not a side length. As the distance at which the patch is located varies its perceived side to the camera, the image from the calibration that helps us determine the size of the patch must be at the distance at which we want to perform the attack by placing the patch. The printed patches are around 7 centimeters on a side. We also printed a random patch and a white patch for our tests.

IV. Evaluation

A. Static tests

For the static tests, we tried to choose about ten different scenes that are not present in the dataset. For each of these scenes, we tested three different attacks by placing patch A at three different locations in the image. The scenes and the patch A in one of the three positions on each scene can be seen in the mosaic in Fig. 4.

We computed the average patch attack probability for a given scene by taking the overall average of the three average probabilities of having an obstacle obtained for each patch position over 400 model classifications. We compared the results of the attack performed with patch A by reproducing the same measurements with a white patch and a random patch. The test (the result can be seen in Fig. 5) was carried

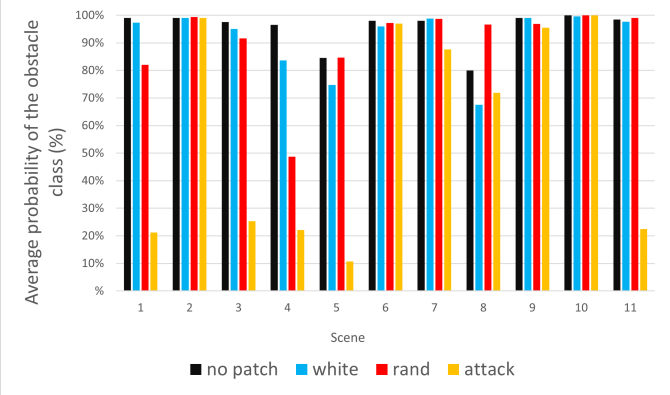

Fig. 5. Results of static attacks

out in one go and an unedited video is available here (Google Drive).

Overall, we had an attack success rate of 42%, which is not far from what the validation curve for patch A indicated (Fig. 3). We quickly observed while testing that static attacks work more or less well depending on the scene that is being attacked. If the background image is a wall or a flat obstacle, the patch works better than if the attacked image contains a vertical obstacle like the legs of a table or a chair. In the test, the first 6 scenes contain obstacles that are flat objects like walls and the success rate of the patch for these 6 scenes is 67%. The other 5 scenes contain vertical obstacles like table or chair legs and the success rate of the patch is only 20%.

One explanation for this difference is that the obstacle class is quite large and groups different clusters of obstacles. One way to see what these different clusters are is to retrieve the activations of the penultimate layer of AlexNet, which contains 4096 features. An image of a given cluster activates some neurons, while an image of another cluster activates others. By performing a principal component analysis on the vectors of 4096 features obtained for each image of the dataset, we can represent the dataset in two dimensions, and then perform an unsupervised learning (k-means clustering) to determine the different clusters. The result of this k-means clustering with a number of clusters equals to 4 can be seen in Fig. 6 and in Fig. 7. The 4 types of images identified by clustering are :

- Images where there is no obstacle
- Images where there are one or more vertical obstacles such as the legs of a table or chair
- Images with obstacles of various shapes such as shoes or keys
- Images where there is a flat and wide obstacle like a wall

An attack consists in moving a point from an obstacle cluster to the ”free” cluster. We notice that the barycenter of the flat obstacle cluster (”blocked2” cluster) is closer to the barycenter of the clear path cluster than the others obstacle barycenters. This is why obstacles like walls are more easily attacked.

We also notice that the average attack vectors do not point directly to the barycenter of the ”free” cluster, but to an area not defined by the model. The functioning of adversarial

Fig. 6. Results of the k-means clustering: the first row is composed of images from the ”free” cluster, the second from the ”blocked0” cluster, the third from the ”blocked1” cluster and the fourth from the ”blocked2” cluster

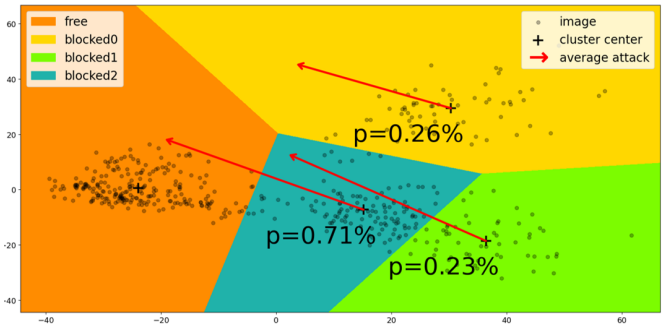

Fig. 7. k-means clustering to detail the obstacle class

patches can be explained by the existence of shadow zones in the model, which has only learned to recognize a small number of images from the infinite number of possible images [11].

B. Dynamic tests

We did some dynamic tests, without changing the classification frequency of the JetBot compared to a normal movement, but making it stop if it detects an obstacle (it should normally turn left and continue its route). We only chose the attack scenes in order to have convincing results and therefore we

mainly attacked walls and flat obstacles. The uncut videos of the attacks are available here (Google Drive).

V. Conclusion

Finally, we succeeded in the main objective of the project which was to create adversarial patches that can fool the JetBot model and make the robot run into an obstacle when a patch is placed on it. We believe that we have shown that a physical adversarial patch attack of an image classifier can be achieved provided that a large number of parameters are taken into account, which can be intrinsic to the camera (digital noise, radial distortion...) or extrinsic (angle of view of the camera on the patch, external luminosity, light reflections on the patch...). These parameters, which can vary over time, make the attacker's task much more difficult, which is reassuring as to the robustness of image classifiers to this type of attack.

However, we have shown that an attacker can target weak points in the model, such as classes that are more easily attacked than others. By patching only on walls, the attack against the JetBot classifier works 2 times 3.

References

[1] C. Lu and X. Tang, "Surpassing human-level face verification performance on LFW with gaussianface," *CoRR*, vol. abs/1404.3840, 2014. [Online]. Available: http://arxiv.org/abs/1404.3840

[2] Reuters, "Tesla contraint de rappeler plus de 360.000 véhicules aux États-unis," *Le Figaro*. [Online]. Available: https://www.lefigaro.fr/societes/tesla-contraint-de-rappeler-plus-de-350-000-vehicules-aux-etats-unis-20230216

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[4] JetBot. (2023) Jetbot index. [Online]. Available: https://jetbot.org/master/index.html

[5] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," *CoRR*, vol. abs/1712.09665, 2017. [Online]. Available: http://arxiv.org/abs/1712.09665

[6] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples," *CoRR*, vol. abs/1707.07397, 2017. [Online]. Available: http://arxiv.org/abs/1707.07397

[7] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1528–1540. [Online]. Available: https://doi.org/10.1145/2976749.2978392

[8] S. Thys, W. V. Ranst, and T. Goedemé, "Fooling automated surveillance cameras: adversarial patches to attack person detection," *CoRR*, vol. abs/1904.08653, 2019. [Online]. Available: http://arxiv.org/abs/1904.08653

[9] D. Karmon, D. Zoran, and Y. Goldberg, "Lavan: Localized and visible adversarial noise," *CoRR*, vol. abs/1801.02608, 2018. [Online]. Available: http://arxiv.org/abs/1801.02608

[10] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[11] M. Melis, A. Demontis, B. Biggio, G. Brown, G. Fumera, and F. Roli, "Is deep learning safe for robot vision? adversarial examples against the icub humanoid," *CoRR*, vol. abs/1708.06939, 2017. [Online]. Available: http://arxiv.org/abs/1708.06939