31 - Shaped Adversarial Patches

Huong Quach

Shaped adversarial patches

Huong Y Quach
Department of Cybersecurity
Old Dominion University
5115 Hampton Blvd, Norfolk, VA 23529
hquac001@odu.edu

Abstract

In recent years, the development and deployment of computer vision models have become widespread, with applications ranging from autonomous vehicles to security systems. Among these, object detection algorithms like YOLO are particularly significant due to their real-time performance and accuracy in identifying and localizing objects within an image. However, the robustness of these models is increasingly challenged by adversarial attacks, which are deliberate manipulations designed to deceive the model's predictions.

In this paper, I present an approach to advancing the deception capabilities of adversarial patches, specifically targeting YOLO-based person detectors. The objective is to design and implement shaped adversarial patches that can be applied directly over a person, effectively lowering the confidence scores of the detector and ultimately fooling the system into failing to recognize the person altogether.

This work builds on existing research in adversarial machine learning, where the creation of adversarial patches has been shown to significantly undermine the reliability of computer vision models. By exploring new patch shapes and configurations, I aim to enhance the effectiveness of these patches, making them more adaptable and capable of bypassing detection algorithms. The results of this research could have profound implications for the security and reliability of AI systems in real-world environments, where adversarial attacks pose a growing threat.

Figure 1: Comparing control patch, initial star patch, star patch after 99 epoch, and circular patch. Patch initialized from a star shape and trained for 99 epochs outperforms the control patch. Proving the concept that initializing from a shaped patch may yield better detection loss.

Introduction

Machine learning, a subset of artificial intelligence, has revolutionized the way computers interpret and analyze vast amounts of data. In particular, computer vision algorithms, which enable machines to perceive and understand visual information from the world, have seen remarkable advancements. Among these, the YOLO algorithm [1] stands out as a highly efficient real-time object detection system. It is capable of identifying and classifying objects within images and video streams with impressive speed and accuracy. However, as powerful as these systems are, they are not without vulnerabilities. Adversarial patches represent a significant challenge in this field [2] [3] [4] [5] [6]. These are carefully crafted patterns designed to deceive machine learning models, specifically targeting object detection algorithms like YOLO. By subtly altering an image, adversarial patches can cause the system to misidentify or completely overlook objects, highlighting the ongoing need for robust defenses against such exploits.

Adversarial patches exploit the vulnerabilities in machine learning models by introducing noise or patterns that are imperceptible or seem benign to the human eye but can cause significant misclassifications by the model. For example, enabling a person to become undetectable with a printed patch in real life [7]. Adversarial patches achieve this disruption by undergoing a specialized optimization process that fine-tunes the pattern or perturbation to maximize its impact on the model's predictions.

Creating an adversarial patch involves several key elements that work together to craft a pattern capable of deceiving a machine learning model. The process begins with a random pattern or an initial guess for the patch. This patch is placed over a set of training images, which are then fed into the model. These images also have corresponding ground truth labels. The model's responses to these modified images are analyzed. The goal is to observe how the patch affects the model's ability to correctly classify the objects in the images. Using gradient descent, the patch is iteratively adjusted to maximize its disruptive effect. The gradients provide information about how changes to the patch influence the model's output, guiding the optimization process to make the patch more effective at causing misclassification. This process is repeated many times, with the patch being continuously refined to increase its impact on the model. The optimization seeks to make the patch as small and inconspicuous as possible while still causing the maximum disruption to the model's ability to recognize objects [8]. This is done by iteratively adjusting the patch so that when it is placed in an image, it minimizes the confidence of the correct class and maximizes the likelihood of incorrect classifications.

Related work

Interest in adversarial patch attacks has increased in recent years, sparking an arms race between adversarial patches and the development of robust defense mechanisms designed to mitigate their effects. This section will go over notable

works. Then briefly discuss the YOLOv2 that was used for this work.

Adversarial arms race

The “adversarial arms race” refers to the ongoing and rapidly evolving competition between the creation of adversarial patches and the development of defenses against these attacks. As adversarial patches become more advanced and capable of fooling even the most sophisticated models, researchers and practitioners are continuously developing new strategies to protect models from these vulnerabilities. The arms race between adversarial patches and defense mechanisms has led to significant research and developments in recent years. Some notable works that have made an impact include: Brown et al. [9] introduced the concept of the adversarial patch, a small, localized perturbation that can cause a machine learning model to misclassify objects in an image. The study demonstrated that these patches could work across various contexts and even in the physical world, marking a significant leap in the sophistication of adversarial attacks. In 2018 Kevin et al. [11] extended the concept of adversarial patches to the physical world, showing that they are capable of fooling real-world object detectors. By applying adversarial patches to road signs, the authors demonstrated how these attacks could cause a model to misclassify critical objects. In 2020 Xiaoyun et al. [10] introduced a defense mechanism against adversarial patches by utilizing a mechanism called " differential analysis." PatchGuard was a significant advancement in creating more robust defenses by reducing the area of influence that an adversarial patch could have, making it harder for the patches to affect the model’s output.

In 2019 Simen Thys et al. [7] created a system to generate adversarial patches and print them. These patches were specifically designed to deceive person detectors. This work demonstrates the real-world applicability of adversarial attacks beyond just digital manipulation. In their work, they built a patch generator which started with a random initialization. The patches were applied to the INRIA dataset to iteratively train the patch generator. The weights and activation functions were updated at the end of each epoch. The loss function used to train the patch generator is the sum of YOLO detection loss, patch printability, and total variation in the patch colors.

YOLO

In this paper I target the well known YOLOv2 object detector. YOLOv2 [12] is a real-time object detection model that builds on the strengths of its predecessor by introducing several key innovations. It uses a fully convolutional network architecture, known as Darknet-19, which consists of 19 convolutional layers and 5 max-pooling layers to efficiently extract features from input images. YOLOv2 divides the input image into a grid and predicts bounding boxes, object confidence scores, and class probabilities for each grid cell. The model incorporates anchor boxes, allowing it to predict bounding boxes of varying aspect ratios and sizes more effectively. These anchor boxes are optimized using k-means clustering, ensuring they are well-suited to

the objects in the dataset. YOLOv2 also employs batch normalization across all convolutional layers to stabilize training, and a pass-through layer that combines high-resolution features with coarse features, improving the detection of smaller objects. Additionally, the model is trained using a multi-scale approach, making it robust to different input resolutions. The result is a highly efficient and accurate object detector capable of processing images at up to 67 frames per second, suitable for a wide range of real-time applications.

Methodology

This work was conducted using three different machines: an Alienware desktop running Ubuntu 24, a Legion Shm 5 laptop with Windows 10, and a Jetson Nano. The Alienware machine was equipped with an Intel Core i7 CPU, an NVIDIA GeForce RTX 3070 Ti graphics card, and 1TB of storage, providing a powerful setup for intensive computational tasks. The Legion Shm 5 laptop featured an AMD Ryzen 5 7000 series CPU, an NVIDIA GeForce RTX 3050 graphics card, and 16 GB of RAM, offering a balanced performance for development and testing. The Jetson Nano, with its 128-core NVIDIA Maxwell GPU, quad-core ARM Cortex-A57 processor, 4 GB of 64-bit LPDDR4 memory, and 128GB microSD storage, ran on Linux for Tegra (L4T), making it ideal for mobility.

For software development and code execution, PyCharm was chosen as the integrated development environment (IDE) due to its robust features that streamline the coding process. Its capabilities, such as on-the-fly error checking and comprehensive support for Python libraries, made it particularly well-suited for machine learning and computer vision tasks. The primary framework used to build the algorithms for the experiment was PyTorch, which facilitated the training and fine-tuning of models. Conda was used as the environment manager and interpreter, ensuring a consistent and reproducible development environment across different platforms. The EAVISE GitLab repository of adversarial-yolo provided the code and the neural networks used for the experiments. The patches were initially run on the Alienware machine, and later run on the windows machine, then deployed against yolov2 on the jetson nano.

The goal of this work is to generate printable shaped adversarial patches that can be used to fool person detectors. As discussed earlier, Simen Thys et al. [7] already showed a system to generate printable patches that are capable of fooling a YOLO person detector. Most work done on adversarial patches, including Simen & Wiebe, use a square patch of various sizes and transformations.

In this work I present two studies. The first study examines the impact on detection loss if the patch shapes are varied and using various striped patches. The second study uses some of these patches as seeds to continue training the adversarial patch generator. I experiment with different shaped patches in order to test their effectiveness at deceiving YOLOv2. I began by selecting their top performing patches as seeds to further train the adversarial patch creator to generate shaped patches. I then

tried seeding with various shaped patches to run through the optimization process.

Dataset Used for this work

This work uses a dataset from INRIA (*Institut de national de research en sciences and technologies* based in France). It consists of over six hundred images with over 2000 people. I followed the same method as Simen Thys et al. [7], my workflow is as follows: I first run the target person detector over the INRIA dataset of images. This yielded bounding boxes showing where people were in the image as detected by YOLO. On a fixed position relative to these bounding boxes, a patch is applied to the image. The resulting image is then fed, in batches of 20 back into the detector. I measured the score of the persons that are still detected. Using back propagation through the entire network, the optimiser then changes the pixels in the patch further in order to fool the detector even more. In my tests, I used images from my family photo album. Each image in the dataset is unique, differing in the number of people present, the background scenery, and the extent to which objects or individuals are obscured. This diversity ensures a comprehensive evaluation of the adversarial patches' effectiveness across a range of real-world scenarios. Depicted in figure 2 are Crops 001007, 001097, 001267, and 001842.

Figure 2

Results

For the first study eight patches were generated and tested against YOLOv2 to measure the detection confidence. These patches are illustrated in figure 3. The code from Simen and Wiebe [7] were used to apply these patches to the INRIA images. The dataset was divided into 31 batches with 20 images in each batch. The patches depicted in column two of figure 3 were run as is with no further training. Note that the first patch (adv) is from work by Simen [7]. The other patches are based on their work with other image processing. The striped images were downloaded from the web. The base confidence scores are shown in column three. The table also gives non printability scores and total variation loss metrics. The test on each patch ran for about two minutes, as shown in column six.

Patch ID	Patch Image	Yolo Confidence Score	NPS LOSS	TV LOSS	RunTime (seconds)
Adv		0.6524	0.00062	0.24409	108.82
AdvCircle		0.7102	0.00055	0.24108	113.60
Circle2		0.7364	0.00051	0.26164	108.52
star		0.7680	0.000354	0.24281	111.25
heart		0.7608	0.000497	0.19599	108.71
Stripes_3D		0.8231	0.00056	0.20855	108.57
Stripes_angle		0.8188	0.00065	0.32014	107.25
Stripes_checker		0.8260	0.00071	0.363272	106.61

Figure 3

This test examines circular patches (advcircle, circle2), irregular shaped patches (star and heart), and striped patches (stripes_3D, stripes_angle, stripes_checker). For each patch the YOLOv2 confidence scores were logged for every image into a text file. The code was modified to log the confidence scores for all images. The text files were used to plot the distributions. Several of the distributions are plotted in figure 4. A sample of the INRIA with the corresponding patch is also given in figure 4. Note that the distribution given in figure 4 represents all images in the dataset with the corresponding patches applied.

As it turns out, the patch shape or content variations did not seem to improve the detector loss over the original square patch. The histograms for each patch are summarized in whisker plots in figure 4. Note that several of the patches are capable of lowering the confidence scores to 20-40%, as seen on figure 4. This has grave implications for vision systems used to prevent self-driving cars from hitting pedestrians.

Figure 4: Graphs show detailed histograms for yolo confidence per each patch over the entire dataset with patch applied. Patch images are samples of the patched images fed to yolov2

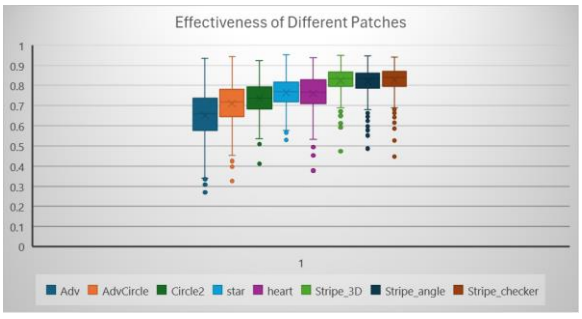

Figure 5: whisker plot of the yolo confidence distribution for the tested patches

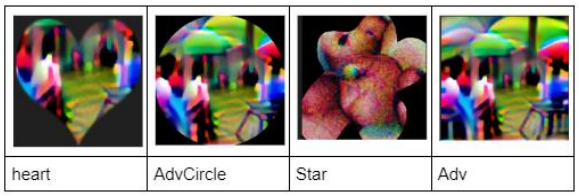

Figure 6: patches before training

Figure 7: Patches after 20 epochs

Based on figure 6 several patches were selected as seeds, or initialization points, to further train the adversarial patch maker. Amongst the selected patches are: heart, star, and AdvCircle depicted in Figure 7. The Adv patch was also run for a comparison. Each patch was trained for 20 epochs consisting of 31 batches in each epoch. Each batch contained 20 images from the INRIA dataset. The adversarial code is structured to update weights after each epoch.

Figure 6 shows the patches before training. Figure 7 depicts the patches after 20 epochs of training. Note that in the first case the *heart* is initialized as a well defined shape. After 20 epochs of training it morphs to fill in the external areas such that only a faint outline of the *heart* geometry is still visible. Likewise in the *star* patch, after 20 epochs of training, the outline of the *star* patch is barely visible. The *adv* patch is run as a control to compare the detector loss results.

The code was modified to add coefficients for each of the three cost components in the cost function. The loss function in [7] can be rephrased for simplicity as follows.

$$Total\ Cost = (alpha * NPS) + (beta * TV) + (charlie * yolo\ loss)$$

Alpha is the weight for the non printability score, beta is the weight for the color variation, and charlie is the weight for detector loss.The adversarial patch trainer minimizes the total cost using the stochastic gradient descent with the *ADAM* optimizer. The test emphasizes *yolo loss* in the adversarial patch training process. Figure 8 shows the training weight values used for this test compared to the original values. Note that the detector loss weight is increased by 100% while the color variation is reduced and the detector loss doubled.

Cost component coefficients	Original values from [7]	Values used for this test
Non printability score weight (alpha)	0.01	0.01
Color variation weight (beta)	2.5	0.5
Detector loss weight (charlie)	1	2

Figure 8: weights for adversarial patch trainer cost function

It is thought that seeding the adversarial patch model with previous runs would improve the training speed of the adversarial patch generation model. Figure 9 shows a comparison of two 20 epoch runs starting with different seeds. For this comparison the *heart* shape seed patch is compared to the *adv* reference patch from [7]. Figure 9 shows the three metrics in the

loss function as well as the total loss (bottom left). It is apparent from the detector loss that the *adv* patch is a better starting point to continue training than the heart shaped patch (top left). But the nps score shows that the heart patch is easier to print (top right). The color variation for both patches are about the same (bottom left). The total loss shows that the adv patch is more effective than the shaped heart (bottom right). A short 20 epoch run is a good tool to determine which starting patch is best to use for continuation of the training for the adversarial patch maker model.

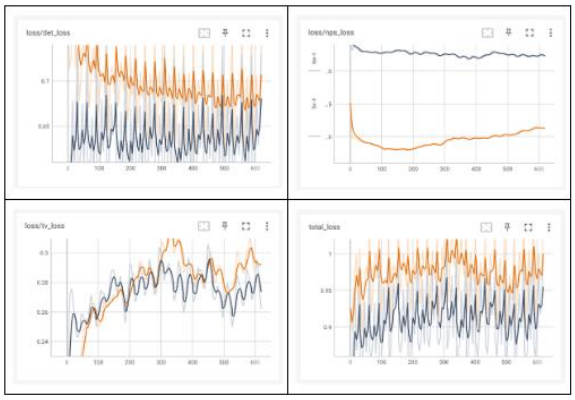

Figure 9: Comparing loss metrics between heart (orange trace) and control adv patch (blue trace).

The star patch was studied more thoroughly in a case that is run for 100 epochs. Figure 10 depicts between 20 and 100 epochs. Both runs were initialized with the same seed patch. The resulting patches are similar, but visibly different in the lower left and mid left sections. The detector loss graphs from the tensor board are given in the fourth row. The third row displays the detector loss, which decreased from 74% to 68%. After 100 epochs, the detector loss decreased further to 66%. After 100 epochs there is still perturbation in detector loss. It's possible that more training epochs are needed before the detector loss will converge

Likewise with the total loss function (or cost function) in the last row. After 20 epochs the total loss is 1.02, it appears that after 100 training epochs the total loss decreased to 0.97. It also appears that more training epochs are needed for total loss to converge.

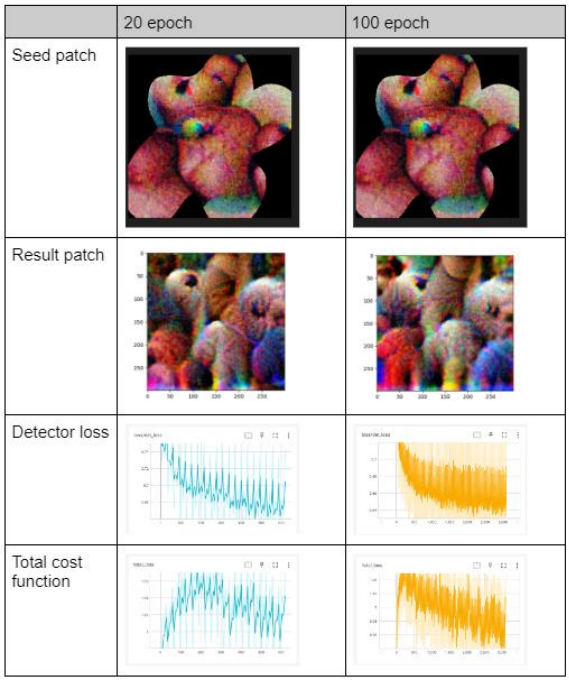

Figure 10: Comparison of star seed patch after 20 epochs and 100 epochs

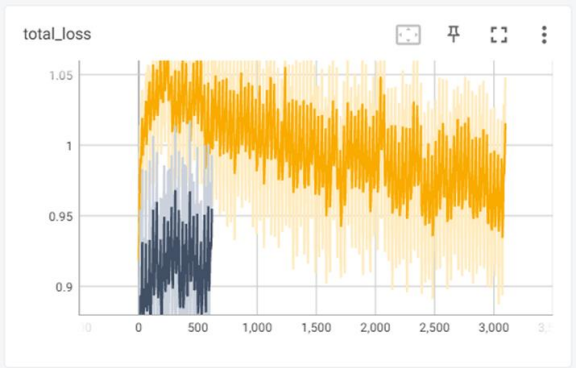

Figure 11: Comparing effectiveness of two different seeds to get to desired solution.

One additional comparison was made to see if time could be saved by starting out with a better seed patch. Figure 11 compares the results of the 20 epoch *adv* patch with the 100 epoch run of *star* patch. The graph shows Blue trace is the total loss of the Adv patch compared to that of the star patch in yellow. It indicates that much training effort may be saved if starting with a better seed patch. As seen in the figure, the *star* patch after 100 epoch is just starting to approach the total loss of the *adv* patch.

Conclusion

This internship began with literature review of computer vision adversarial attacks. Both physical and digital attacks were compared. It was decided to focus on digital attacks because there is no easy way to print adversarial patches to assess their effectiveness in physical attacks.

Reference code from [7] was studied and used to see if the author's patches could be improved upon by using more diverse characteristics. The reference code was modified so that training could be seeded with previously tested patches. This allows study using various seed patches. Some of the characteristics studied include using different shaped seed patches and striped patches. A comparison was done with the author's best performance patch (adv) a series of short training epochs were used to assess the effectiveness of different seed patches to narrow down the selection.

It was found that using a series of short training epochs, it is possible to determine which seed patches will be more effective. The results show that much time could be saved by doing an initial study with short training epochs. An attempt was made to study different shaped patches, but no easy way was found in python to create irregular shaped patches such as stars and hearts. As well as making the background transparent. This is partly because the original code expected RGB (3 dimensional) pixels and did not allow RGBA formats which map into four dimensional pixels. So in this study the shaped patches were placed onto a 300x300 pixel template with varying backgrounds, mostly black to fill the void.

Future work

Given more time there are several areas that could be improved. As mentioned before the majority of the adversarial patch research uses square patches. Effectiveness of irregular shaped patches could be studied in more depth. One of the key performance metrics of the adversarial patch is the misclassification of people in the INRIA database. There was no easy way to determine whether yolo misdetections of a person throughout the entire training set. For future work, the cost function should add a fourth variable that determines misdetections. As always, more time is always useful to run training with a greater number of epochs.

Figure 12: Example of striped patches and heart patch over samples from dataset

Acknowledgements

The completion of this paper has been made possible through the REU program. I'm grateful for the opportunity to learn more about computer vision, adversarial attacks, and gain hands-on research experience. I would like to thank Dr. Peng and Dr. Gladden for mentoring me throughout the summer, providing access to equipment to complete this project, and generously setting time aside for meetings. Additionally, I would like to express my gratitude to the program organizers. This experience has been invaluable in enhancing my knowledge and skills, and I look forward to applying what I have learned in my future endeavors.

References

[1] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. https://www.cv-foundation.org/openaccess/content_cvpr_f

[2] Adhikari, A., Den Hollander, R., Tolios, I., Van Bekkum, M., Bal, A., Hendriks, S., Kruithof, M., Gross, D., Jansen, N., Pérez, G., Buurman, K., & Raaijmakers, S. (n.d.). ADVERSARIAL PATCH CAMOUFLAGE AGAINST AERIAL DETECTION. Retrieved June 10, 2024, from https://arxiv.org/pdf/2008.13671

[3] Brown, T., Mané, D., Roy, A., Abadi, M., & Gilmer, J. (n.d.). Adversarial Patch. Retrieved June 24, 2024, from https://arxiv.org/pdf/1712.09665

[4] Czaja, W., Fendley, N., Pekala, M., Ratto, C. R., & Wang, I-Jeng. (2018). Adversarial examples in remote sensing. https://doi.org/10.1145/3274895.3274904

[5] Du, A., Chen, B., Chin, T.-J., Yee, Law, W., Sasdelli, M., Rajasegaran, R., & Campbell, D. (n.d.). Physical Adversarial Attacks on an Aerial Imagery Object Detector. Retrieved June 3, 2024, from https://openaccess.thecvf.com/content/WACV2022/papers/Du_Physicalpdf

[6] Sharif, M., Bhagavatula, S., Bauer, L., & Reiter, M. K. (2016). Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16.

https://doi.org/10.1145/2976749.2978392

[7] Thys, S., Van Ranst, W., & Goedemé, T. (n.d.). Fooling automated surveillance cameras: adversarial patches to attack person detection. https://openaccess.thecvf.com/content_CVPRW_2019/pape rdf

[8] Ghanem, B., Rosso, P., & Rangel, F. (2018). An Emotional Analysis of False Information in Social Media and News Articles. *Association for Computing Machinery, 1*(1). https://doi.org/10.1145/1122445.112245

[9] Brown, T., Mané, D., Roy, A., Abadi, M., & Gilmer, J. (n.d.). *Adversarial Patch*. Retrieved June 24, 2024, from https://arxiv.org/pdf/1712.09665

[10] Xiang, C., Bhagoji, A., Sehwag, V., & Mittal, P. (n.d.). *PatchGuard: A Provably Robust Defense against Adversarial Patches via Small Receptive Fields and Masking*. Retrieved August 10, 2024, from https://arxiv.org/pdf/2005.10884

[11] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Tramèr, F., Prakash, A., Kohno, T., & Song, D. (n.d.). *Physical Adversarial Examples for Object Detectors*. Retrieved August 10, 2024, from https://arxiv.org/pdf/1807.07769

[12]Redmon, J., & Farhadi, A. (n.d.). *YOLO9000: Better, Faster, Stronger*. Retrieved May 17, 2024, from https://arxiv.org/pdf/1612.08242

[13] *Adversarial patch attacks on self-driving cars*. (2022). Blog.neater-Hut. https://blog.neater-hut.com/adversarial-patch-attacks-on-self-driving-cars.html

[14]Liu, X., Yang, H., Liu, Z., Song, L., Li, H., & Chen, Y. (n.d.). *DPATCH: An Adversarial Patch Attack on Object Detectors*. Retrieved August 10, 2024, from https://arxiv.org/pdf/1806.02299