Trustworthy Machine learning
Final Project Report

By Pallaw Kumar and Jason Li

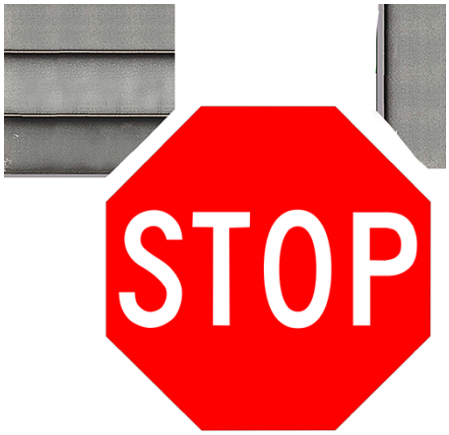
Autonomous driving system overview

An autonomous driving system, also known as a self-driving car or autonomous vehicle (AV), is a technology that allows a vehicle to operate and navigate without direct human input. It combines various sensors, software, and computing power to perceive the environment, make decisions, and control the vehicle's movements.

However, the development of AVs has encountered various challenges, including the vulnerability of computer perception systems. Accurately detecting and classifying objects in real-time is critical for AVs. However, computer perception systems can face challenges when dealing with complex or ambiguous scenarios, such as occluded objects, unusual object appearances, or identifying small objects at a distance. Improper object detection or misclassification can impact the decision-making process and compromise safety.

The symbolic Importance of stop signs

Adversarial attacks on stop signs, and more broadly on traffic signs, are a compelling and important research project with several notable benefits. Such research helps to uncover vulnerabilities in autonomous driving systems and contributes to the development of more robust and secure AI models. It helps us identify vulnerabilities in autonomous driving systems, enhance their safety and security, evaluate their performance, address ethical considerations, and advance the field of adversarial machine learning. By exploring and mitigating these vulnerabilities, we can work towards the widespread adoption of reliable and trustworthy autonomous vehicles, ensuring a safer and more secure future on the roads.



Problem statement

While autonomous driving systems heavily rely on visual perception to recognize and interpret traffic signs, they are susceptible to misclassification or misinterpretation when exposed to manipulated signs. By strategically placing overlapping images on traffic signs, an attacker can potentially exploit the vulnerabilities in the perception algorithms of autonomous vehicles, leading to incorrect identification or misinterpretation of the signs.

The objective of this project is to investigate the vulnerability of autonomous driving systems to a specific attack known as the patch attack. The patch attack is a physical black box attack that involves adding overlapping images or patches onto traffic signs with the intention of deceiving autonomous vehicles.

The objective of this project is to construct a confidence surface for a specific patch attack, such as the Goldfish attack, and propose an optimization algorithm to identify the optimal attack. The confidence surface represents the relationship between the patch characteristics (e.g., size, shape, position) and the likelihood of successful manipulation, as determined by the impact on the perception system of an autonomous driving vehicle.

The specific challenges to be addressed within this project are as follows, a detailed explanation of those challenges can be found in appendix A.

- Confidence Surface Construction
- Surface Smoothness Analysis
- Optimization Algorithm Development
- Attack Evaluation Metrics
- Experimental Validation

Resources limitations

Due to both time and computational power constraints, the project will utilize the ResNet-50 model as the image classification model. Ideally, the Safebench model can be deployed in future development and test.

Methods and Technical contributions

- **Confidence Surface Construction:** construct a confidence surface by iteratively applying an attack patch (size 32 by 32) over a stop sign image (size 224 by 224) and extracting the confidence level of the model prediction. The results are as the following:

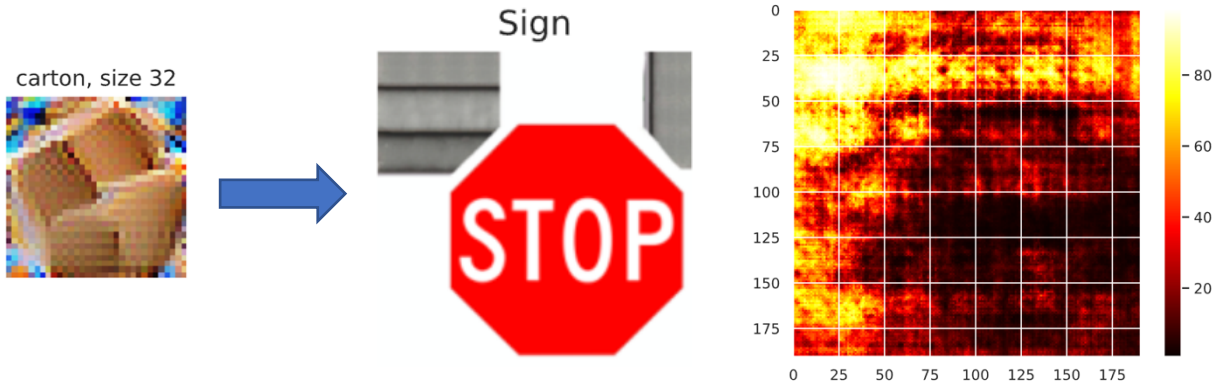


*Figure 1 Attack illustration and Heat Map of **Carton 32** Attack Confidence Output*

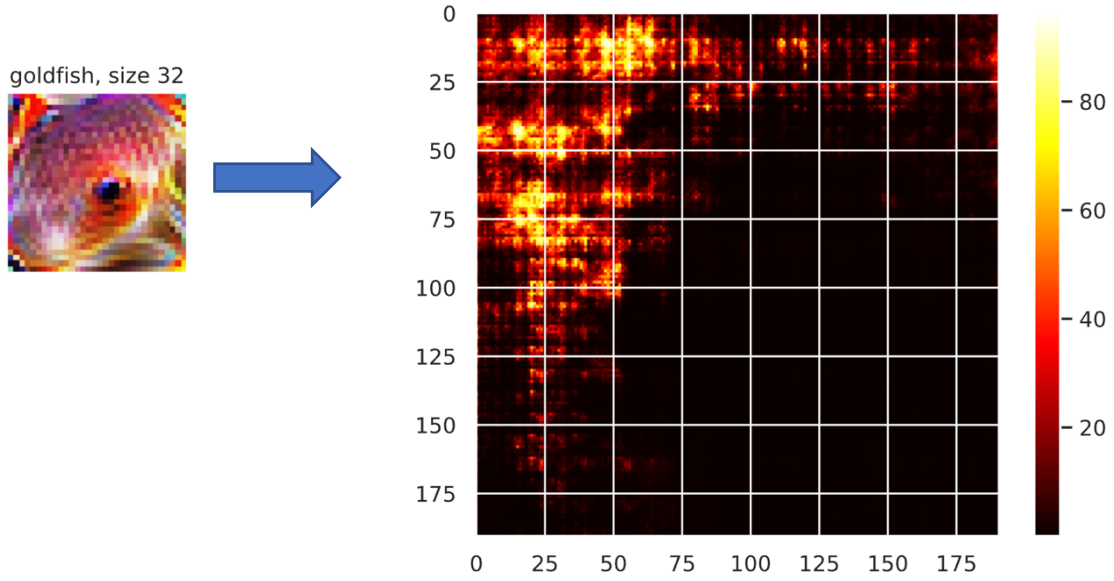


*Figure 2 Heat Map of **Goldfish 32** Attack Confidence Output*

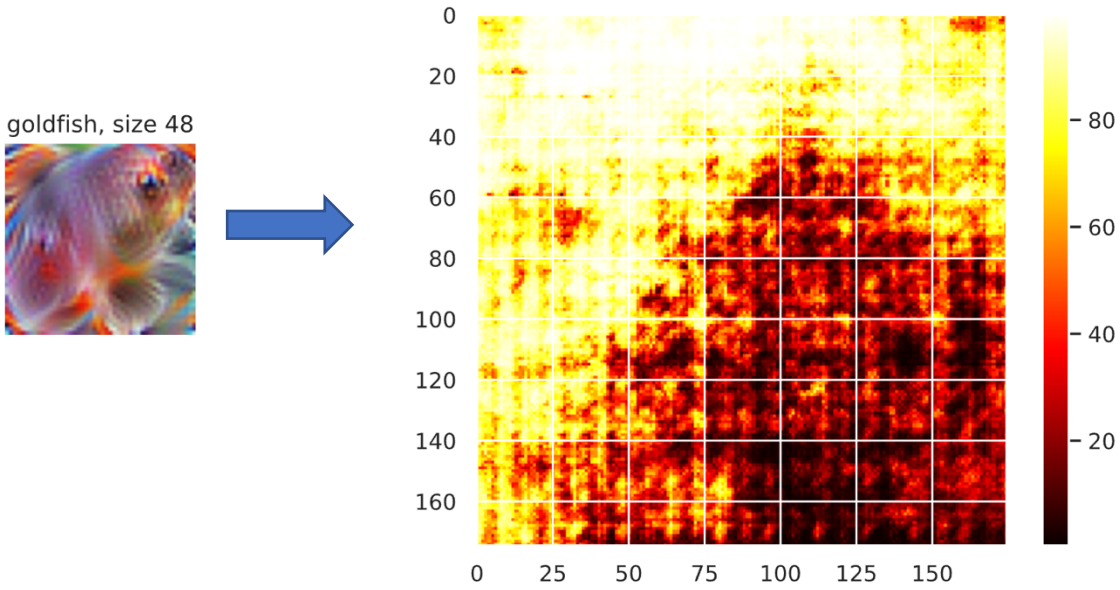


*Figure 3 Heat Map of **Goldfish 48** Attack Confidence Output*

- **Surface Smoothness Analysis:** from the constructed confidence surface heat map, we can observe that the surface is not smooth. Contrary to what we have seen in the project presentation, the response surface algorithm will not work here. We need to find some other techniques for finding optimal attack position.
- **Optimization Algorithm Development:** Based on the observation that high-value points on the confidence surface tend to cluster together in the heat map, we aim to propose a statistical method to speed up finding the optimal attack position. The objective is to

leverage the clustering patterns to guide the optimization algorithm and reduce the search space for identifying the optimal attack. The method should effectively reduce the search space by focusing on the most promising regions, enabling faster convergence without sacrificing the quality of the optimal attack configuration.

The proposed method works as the following:

1. Random Initial Attack Position: Select an initial attack position randomly on the confidence surface. This position will serve as the starting point for the optimization process.
2. Neighborhood Sampling: Randomly sample points within the neighborhood of the initial attack position. The neighborhood can be defined based on a predefined distance or radius around the initial position. (*Hyperparameter*: distance/radius, number of neighborhood points being sampled)
3. Confidence Value Comparison: Compare the confidence value of the initial attack position with the confidence values of the sampled points. Determine whether each sampled point has a higher or lower confidence value compared to the initial position.
4. Move or Stay Decision: If a sampled point has a higher confidence value than the initial position, move to the new position. Update the current position to the sampled point with the higher confidence value. If a sampled point has a lower confidence value, record the current position with the confidence level, and start another initialization.
5. Iterative Sampling and Decision Making: Repeat the process of sampling points within the neighborhood, comparing confidence values, and making move or stay decisions iteratively. Continue until a stopping criterion is met, such as reaching a maximum number of iterations or achieving a desired level of confidence value improvement. (*Hyperparameter*: stopping criterion)

We have successfully implemented a function with the proposed method in mind. Having all the hyperparameters as arguments in our function provides flexibility and allows for experimentation to find the optimal values for different scenarios. The code can be found in a Google Colab Notebook.

- **Attack Evaluation Metrics and Experimental Validation**: We use target class confidence and the amount of time the algorithm used as the evaluation matrix to test the effectiveness and speediness of the method. We think confidence above 40 is big enough to guarantee a successful attack. Our algorithms can find a good attack position with over 80 confidence value quickly, usually within 30 seconds on the **Goldfish32** attack. With the hyperparameters we designed in the algorithm, we can tune the performance of this algorithm and find the hyperparameters for the optimal performance, with respect to different target class and image.

Due to time limitation, we did not do extensive tests. However, based on user experience, we can say that it is very effective and fast compared to brutal force method. If given the time and the need, we'd like to do a comprehensive experimental validation, utilize representative datasets and perception systems to evaluate both speed and performance of

this algorithms. Compare the results with baseline attacks to demonstrate the superiority of the proposed approach.

```
Iteration Number 0.
Max confidence value 0.23391307331621647 patch location in image: H=18 W=131. Number of calls required 21.
Iteration Number 1.
Max confidence value 66.77986979484558 patch location in image: H=102 W=24. Number of calls required 47.
Iteration Number 2.
Max confidence value 66.77986979484558 patch location in image: H=102 W=24. Number of calls required 57.
Iteration Number 3.
Max confidence value 71.05977535247803 patch location in image: H=96 W=64. Number of calls required 87.
Iteration Number 4.
Max confidence value 71.05977535247803 patch location in image: H=96 W=64. Number of calls required 127.
Iteration Number 5.
Max confidence value 71.23673558235168 patch location in image: H=111 W=88. Number of calls required 166.
Iteration Number 6.
Max confidence value 84.80456471443176 patch location in image: H=42 W=56. Number of calls required 206.
Iteration Number 7.
Max confidence value 84.80456471443176 patch location in image: H=42 W=56. Number of calls required 246.
Iteration Number 8.
Max confidence value 84.80456471443176 patch location in image: H=42 W=56. Number of calls required 266.
Iteration Number 9.
Max confidence value 84.80456471443176 patch location in image: H=42 W=56. Number of calls required 296.
Execution time: 28.743507385253906 seconds
```

Figure 4 Algorithm Demo

- **Insights and lesson learned:**

1. *Universal Vulnerability*: We find that Resnet-50 is a very vulnerable model under patch attack. We suspect all deep learning models are vulnerable to patch attack given that the underling training methods are similar: Gradient descent and backpropagation. Without dramatic change in the underlying mechanism, we suspect the robustness will improve.
2. *Higher Dimension Extension*: Our method could be easily transferred to higher dimension. However, we suspect that something in higher dimension could change the calculations in many ways. So, we will be humble here and say we think more experiments are required to solve the same problem in higher dimension.
3. *The tradeoff between speed and perfection:* If you want the highest confidence value, whole map search might be the only way, given that no access to the model gradients. If you want to find a good enough attack configuration, our algorithm might help.
4. *Please see the section:* ***Potential improvements in the Future*** for more insights.

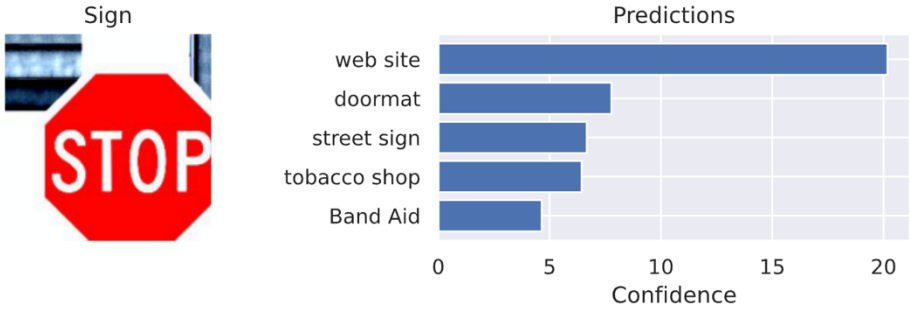
Surprising points and fun facts

Surprising point 1: Under Resnet-50, we found that: before attack, the model predicts incorrectly. After FGSM attack, the model predicts correctly.

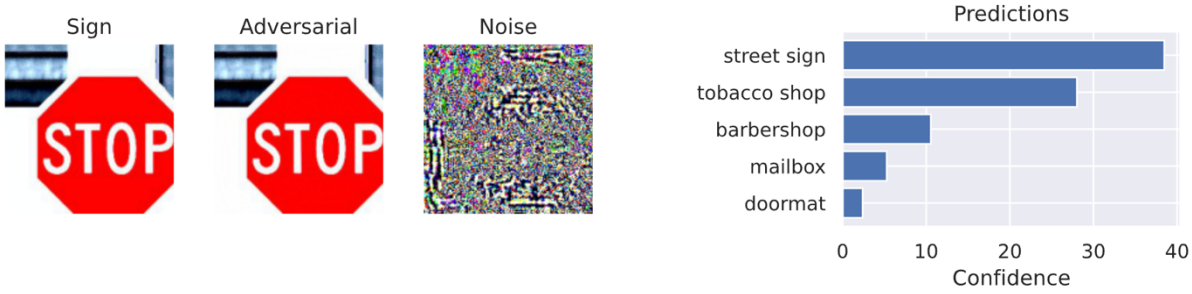
It is possible that the FGSM attack was able to improve the model's predictions by making the input image more like the training data.

It is also possible that the FGSM attack was able to improve the model's predictions by simply making the input image noisier. Noisy images can be more difficult for humans to classify, so it is possible that the FGSM attack was able to make the input image more difficult for the model to classify as well. This would result in the model predicting incorrectly before the attack and correctly after the attack.

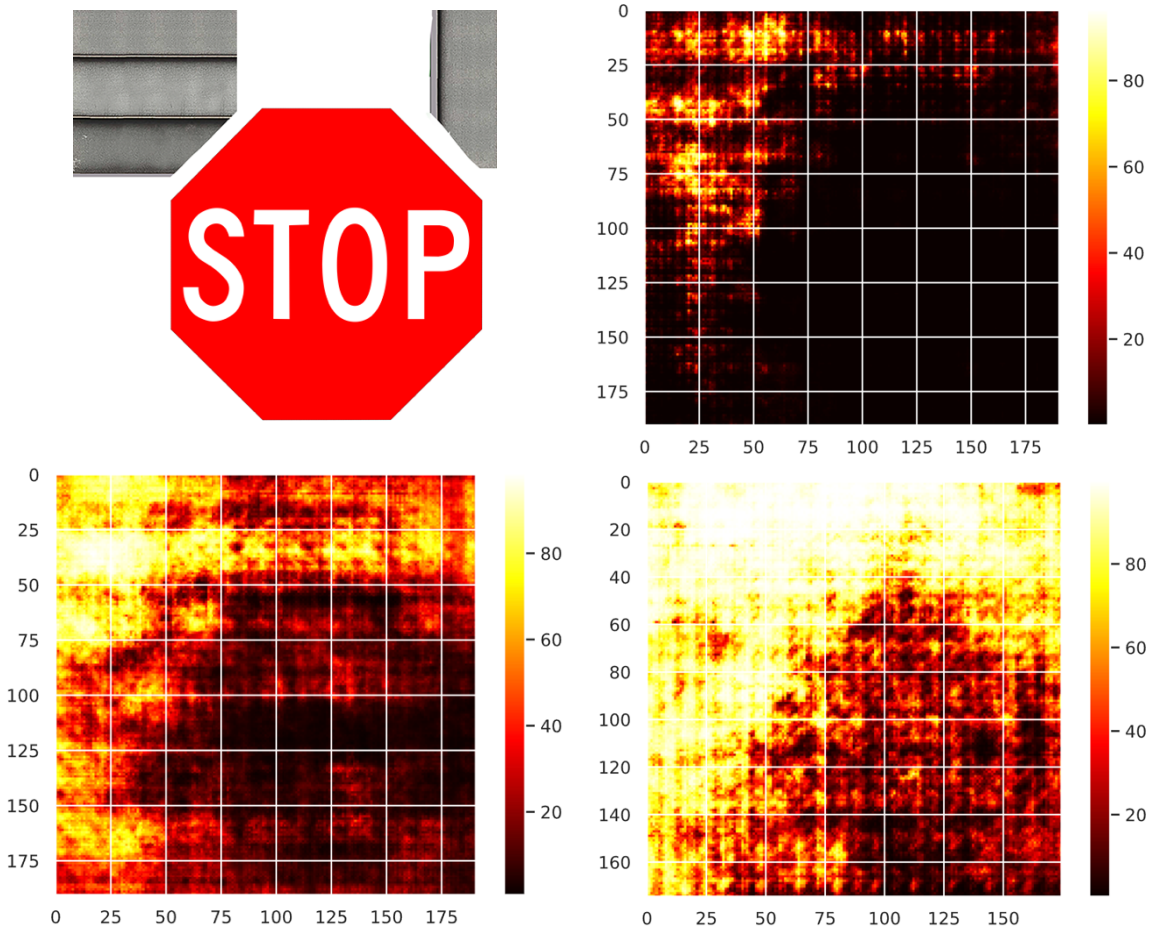
Before attack:



After FGSM attack:



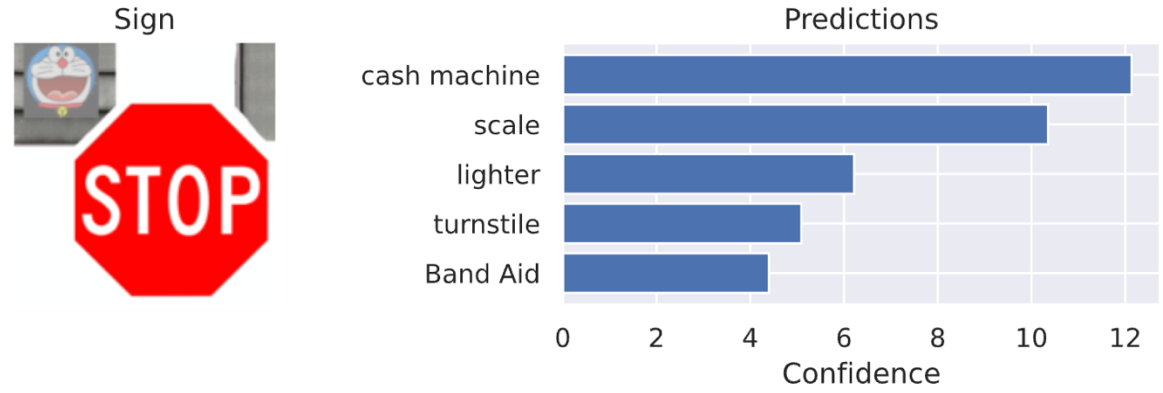
Surprising point 2: as we have seen in the previous section, the attack on the position around stop signs seems to be most successful.

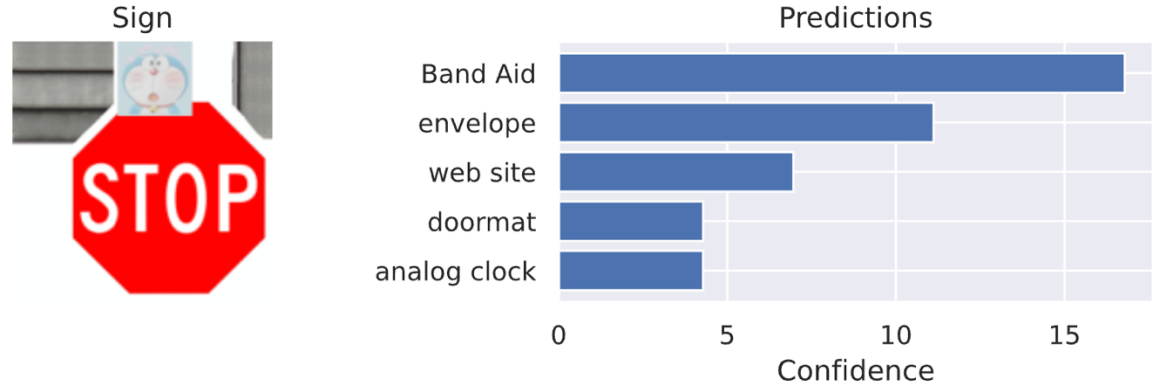


Why is that? We don't really know. We hypothesize that some attention mechanism analysis of Resnet-50 might be helpful.

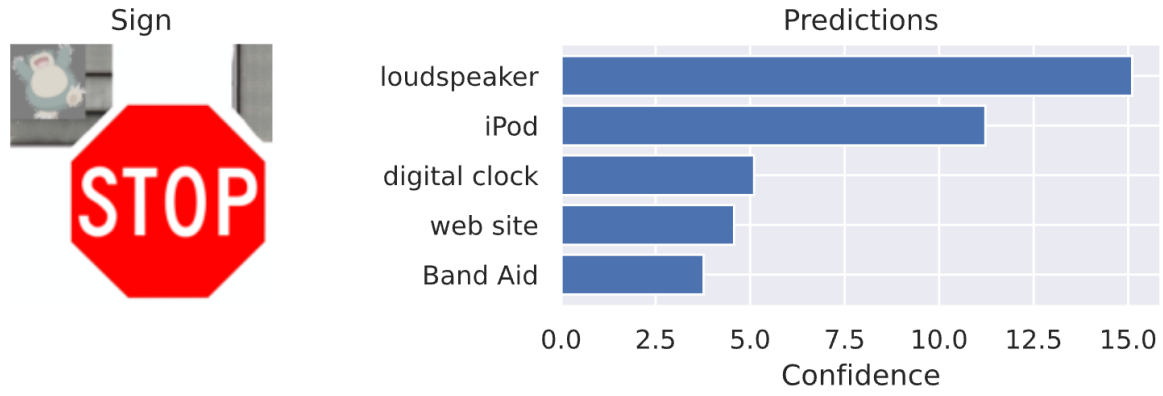
Surprising point 3:

Does anyone think Doraemon is a cash machine or a Band Aid?





Is Snorlax a loudspeaker or an iPod?



Potential improvements in the Future

Sensitivity to Initial Attack Position: We may observe that the choice of the initial attack position has a significant impact on the optimization process. In some cases, starting from a position with a high confidence value can lead to faster convergence, while starting from a position with a low confidence value may require more iterations to achieve significant improvements. This insight can guide us in developing strategies to choose initial attack positions more effectively.

Neighborhood Size and Sampling: The size of the neighborhood and the number of points sampled within it can greatly influence the exploration-exploitation trade-off. A smaller neighborhood with fewer samples may result in faster convergence but risk getting stuck in local optima. On the other hand, a larger neighborhood with more samples can help explore a broader search space but might require more iterations to converge. Analyzing the relationship between neighborhood size, sampling, and the rate of improvement can provide insights into finding an optimal balance.

Stopping Criterion and Convergence: The stopping criterion we defined for the optimization process is an important factor in determining when to end the iterations. Analyzing the convergence behavior of the algorithm can help us understand how many iterations are typically required to achieve the desired level of confidence value improvement. This insight can guide us in setting an appropriate stopping criterion based on the problem at hand.

Robustness and Generalization: Testing the method on various datasets or scenarios can reveal its robustness and generalization capabilities. Evaluating how well the method performs across different types of data or models can provide insights into its effectiveness and limitations. We may discover specific data patterns or model characteristics where the method excels or struggles, helping us understand its applicability in different contexts.

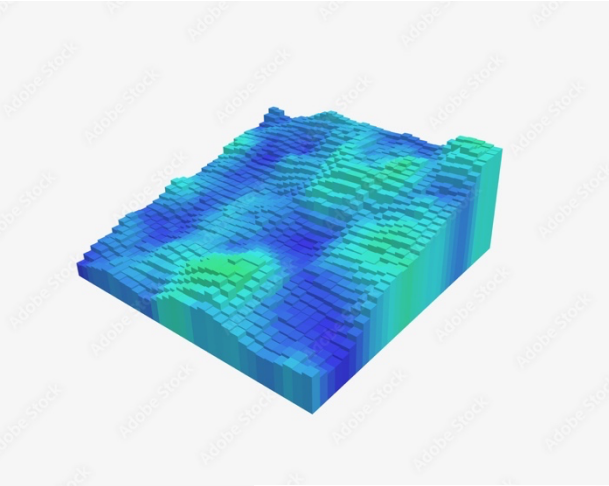
Hyperparameter Tuning: The hyperparameters in our method, such as the distance/radius, number of samplings, and number of iterations, can significantly impact the optimization process. Conducting experiments with different hyperparameter settings and analyzing their effects on the results can help us fine-tune the method and improve its performance. Consider using techniques like grid search or random search to systematically explore the hyperparameter space.

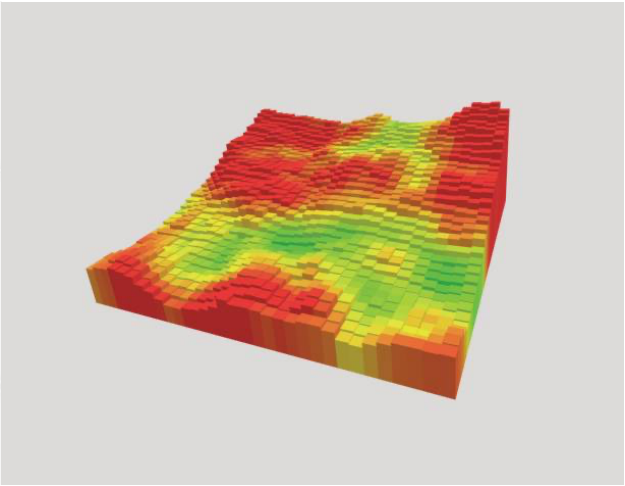
Visualization and Interpretation: Visualizing the optimization process, such as plotting the confidence values over iterations or mapping the attack positions on a surface, can provide valuable insights. It can help us understand the behavior of the method, identify patterns, and interpret the results more effectively.

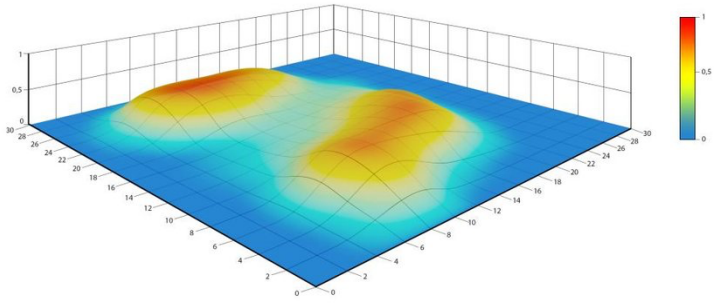
If given the opportunities, we'd like also to improve our deliver by implement the following additional work:

- We'd like to do Hello Katie attacks on stop signs.

- We'd like to do 3D heat maps.



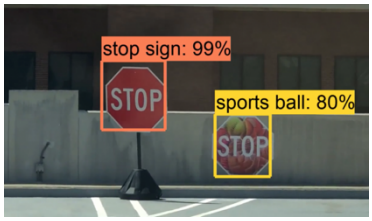




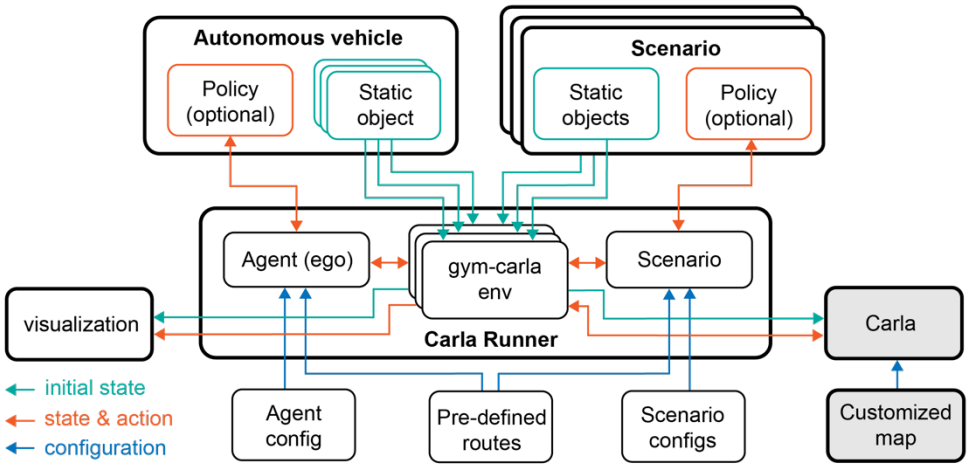
- We'd like to attack more realistic stop signs.

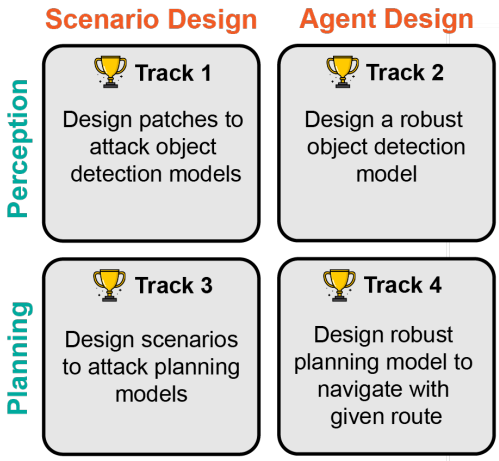
Human: 100.0 % stop sign
Machine: 99.7 % stop sign

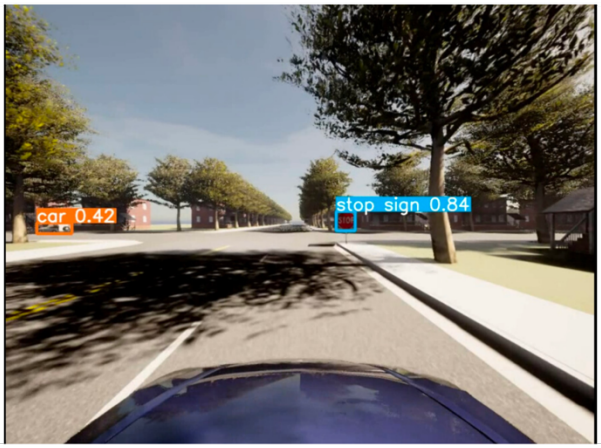
Human: 100.0 % stop sign
Machine: 0.9 % stop sign



- We'd like to have access to Safebench and Carla







Most important, we'd like to have professional guidance and feedback, a place to show our work, and a great team.

Appendix A

- Confidence Surface Construction: Develop a method to construct the confidence surface by systematically varying the patch characteristics and evaluating their impact on the perception system. This involves designing experiments to capture the perception system's response to different patch configurations, collecting relevant data, and analyzing the results to derive the confidence surface.

- Surface Smoothness Analysis: Investigate the smoothness of the constructed confidence surface. Assess the continuity and regularity of the surface to determine if it exhibits a smooth behavior. If the surface is smooth, further analysis and optimization can be performed to identify the optimal attack.

- Optimization Algorithm Development: Propose an optimization algorithm tailored to the smooth confidence surface. The algorithm should efficiently explore the surface to identify the optimal patch characteristics that maximize the likelihood of successful manipulation. Consider both local and global optimization approaches to ensure comprehensive search and avoid potential local optima.

- Attack Evaluation Metrics: Define appropriate metrics to evaluate the effectiveness of different attacks. These metrics may include the success rate of the attack, the degree of manipulation achieved, or the impact on the perception system's classification performance. These metrics will be used to guide the optimization algorithm in identifying the optimal attack configuration.

- Experimental Validation: Validate the constructed confidence surface and the proposed optimization algorithm through extensive experimentation. Utilize representative datasets and perception systems to evaluate the effectiveness and robustness of the identified optimal attack. Compare the results with baseline attacks to demonstrate the superiority of the proposed approach.