2025 Design, Automation & Test in Europe Conference (DATE 2025)
Special Initiative "Autonomous Systems Design"

Generating and Predicting Output Perturbations in Image Segmenters

Matthew Bozoukov
Independent Researcher
matthewbozoukov123@gmail.com

Nguyen Anh Vu Doan
Infineon Technologies AG
anhvu.doan@infineon.com

Bryan Donyanavard
Department of Computer Science
San Diego State University
bdonyanavard@sdsu.edu

***Abstract*— Image segmentation applications are a core component of safety-critical autonomous software pipelines. Sensor data input noise can lead to segmentation output corruption that threatens safety in both DNN- and transformer-based segmenters. Previous work has proposed methods for generating malicious noise to cause DNN- and transformer-based object detection and classification output corruption. We perform the same task for image segmentation applications using genetic algorithms for optimization. We then propose a novel method to predict whether an input image will yield a corrupted segmentation output due to noise. We evaluate the optimal noise generation and corruption prediction on state-of-the-art image segmenters YOLOv8 and DETR. We observe that we can (a) cause segmentation output corruption with noise that is undetectable to the human eye and unrelated to the corrupted region of the image; and (b) predict output corruption due to image noise with over 96% accuracy.**

I. Introduction

The ability of an autonomous cyber-physical system to navigate and interact with its environment hinges on environmental understanding. In this context, image-based object detection and segmentation are common environmental understanding tasks that enable autonomous systems to operate safely in the physical world around humans. Autonomous vehicles must identify objects to navigate traffic, e.g., identify a stop sign, and locate objects to avoid collision, e.g., identify and locate a pedestrian. Neural networks such as YOLOv8 [1] and DETR [2] are utilized in real-time object-detection and image-segmentation applications to achieve such autonomy. Ensuring the safe operations of an autonomous system requires to understand how image input noise and variety play a role in misidentifying objects. In this paper we address the safety concerns that come with image segmentation in autonomous driving and evaluate YOLOv8 and DETR's robustness to noise in input images, i.e., *perturbation*.

We focus on perturbations that achieve a "butterfly effect" as introduced in [3], and address two questions: (1) can we generate perturbations in input images that corrupt the output of state-of-the-art object segmentation neural networks? and (2) can we detect such perturbations in input images? Perturbations should be *tiny*, i.e., undetectable by the human eye, and *unrelated*, i.e., in regions of the image far from the corrupted object. Success in positively answering both questions would imply that object segmentation neural networks cannot prevent malicious attacks solely through the neural network design, but such attacks can be detected before the output is corrupted. We use images from the KITTI dataset [4] to demonstrate and evaluate perturbation.

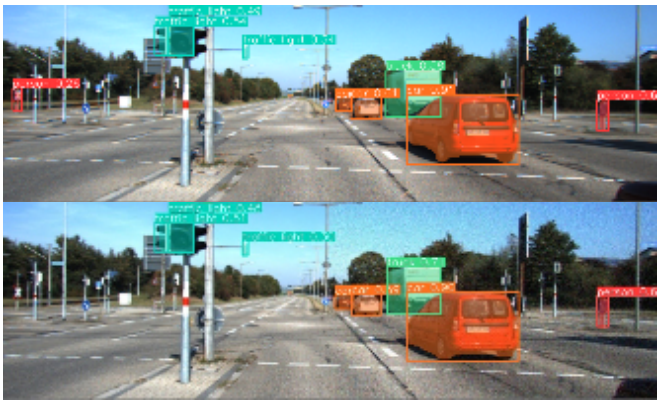

Fig. 1. The first image has no noise, the image below is optimal noise and we can see performance degradation in terms of missing objects and lower confidence scores

Fig. 2. The first image has no noise, the image below is optimal noise and we can see performance degradation in terms of missing objects only however there are more missing objects than in the YOLOv8 figure above

To illustrate the "butterfly effects" of perturbations, we show in Figure 1 two output images from YOLOv8. The top image shows the output with no input perturbation applied. The bottom output has an optimally generated perturbation applied to only the *right side* of the input image. We observe that the noise affects the segmenter's output on the *left side* of the image by not being able to detect the object furthest to the left. Figure 2 shows the same input image segmented with DETR before (top) and after (bottom) perturbation. We observe more objects disappearing as compared to YOLOv8's detection, indicating that transformer-based networks may be less resilient to input perturbations. In this work we make

978-3-9826741-0-0/DATE25/© 2025 EDAA

the following novel contributions:

- We evaluate the ability of two different multi-objective optimization search algorithms to generate perturbations for object segmentation neural networks. We simultaneously solve multiple objective functions to create optimal perturbations by mapping the generated optimal values from each of these functions back to the corresponding input, creating an optimal range of perturbation to apply to a given input.
- We generate and evaluate data-driven CNNs to predict whether an input image, when perturbed, would yield corrupt output from an image segmentation software.

We format this paper in the following way: Section 2 introduces related work and differentiates our work. Section 3 defines the optimization problem. Section 4 presents the genetic algorithms and the results of the multi-objective optimization searches. Section 5 describes the parameters of our CNN design. Section 6 evaluates the CNN, and Section 7 discusses future work.

II. Background and Related Work

The history of adversarial attacks has many angles to look at the specific problem, including adding various types of noise, e.g., Gaussian and salt and pepper noise [5]. Additionally, black-box approaches [6], [7], [8], [9], [10], and white box attacks [10] have also been studied. The attack strategy can vary between an optimization-based approach or a sensitivity analysis. Among approaches similar to ours, FGSM [10] and Carlini-Wagner-attack [11] can be highlighted. FGSM is a white-box attack that can directly access the neural network it affects by modifying the network gradients in order to maximize the loss of the neural network. It requires full control and knowledge of how the neural network operates, whereas our approach does not. The Carlini-Wagner attack utilizes an optimization problem to perform misclassification. It only targets classification and has two objectives: minimizing the perturbation, and maximizing its damage. In contrast, we target in this work segmentation and add a third objective to this attack by requiring the perturbation to be unrelated.

Our approach for generating perturbations is a black-box optimization attack that expands on [3] in two ways: (1) we address object segmentation instead of detection, and (2) we evaluate multiple solvers. Object segmentation networks output image masks, which are more precise than bounding boxes (the output of object detectors). In addition to perturbation generation, we identify whether an image has been corrupted by noise enough so that the segmenter output will be erroneous.

III. Optimization Problem

We formulate our problem as follows: we define an "image segmentation algorithm" as a function f: $R^{L\times W\times 3} \rightarrow B^N$ which takes an image R with length L, width W, and 3 RGB channels, and outputs a vector of prediction masks. Prediction masks have the form (cl,x,y,l,w), where cl denotes the class, x and y denote the current coordinates in the image plane and l and w mean the length and width of the bounidng box around the image. We define cl to to be $\in$ a vector of classes if $cl \in (1, ..., C, *)$ where we define $*$ to be the class of an unknown prediction. If an image is designated with the class $*$, then we do not view this as a valid prediction mask. As a precautionary assumption, $f(img)$ where $img \in R^{LXWX3}$ is a correct prediction. We then generate a perturbation $\delta \in R^{L\times W\times 3}$ which will be applied to $f(img + \delta)$, which might cause misidentified objects (masks), disappearing objects, objects that appear out of nowhere, and shrunken or enlarged image masks. The optimization consists of three objectives:

1) The goal of the first objective function is to make the perturbation as small in magnitude as possible, so as to be undetectable. We measure magnitude in terms of L_2 norm.
2) The second objective function maximizes the corruption of the image, so that the perturbation does as much damage as possible. We consider corruption as disappearing objects or objects appearing that are not in the original image.
3) The last objective function enforces the *unrelated* aspect of the perturbation, in order to reduce predictability of corruption.

A. Unrecognizable Perturbation

Our first objective is to create a perturbation that is unrecognizable relative to human perception by generating the smallest possible perturbation. Since the perturbation is a matrix of values, we can characterize the degree of perturbation through the L_2 norm. In general, a variety of different norms $L_0, L_1, ..., L_\infty$ can be chosen, but we follow suit from [3] and choose the L_2 norm as it is the Euclidean norm and we are dealing with a Euclidean space. Specifically, we are using the $L_{2,1}$ where we take the Euclidean norms of the columns then sum all of them together. This is different from the standard L_2 norm of a matrix where we find the largest singular value of a matrix (square root of the largest eigenvalue of the matrix, multiplied with its conjugate transpose). The standard L_2 norm is computationally intensive, and unnecessary for our purposes.

B. Performance Degradation

The second objective aims to understand how a specific perturbation affects the prediction of an image by maximizing the output distortion. The main focus of this algorithm is measuring the overlap between the output for one image with and without perturbation. We use an intersection-over-union (IOU) algorithm (Algorithm 1) which primarily outlines the difference in a dataset between the detected image and the actual image. In our case the algorithm calculates the difference between the perturbed image detection output and the non-perturbed image detection output. We first loop through the unperturbed image masks, B. For each unperturbed image mask, we loop through all perturbed image masks, B', and compare output classes. Once we find the maximum overlap we add it to the total counter A. After each iteration of

Algorithm 1 Maximize output distortion of image

$A \leftarrow 0$
while *B is a valid class* $\in$ *f(img)* **do**
  $A_0 \leftarrow 0$
  while *B' is a valid class in the perturbed image* **do**
    if *B == B'* **then**
      $A_0 \leftarrow max(IOU(B, B'), A_0)$
    end
  end
  $A \leftarrow A + A_0$
end
return $\frac{A}{\#validboundingboxes}$

the outer loop, we add the difference between each mask to an overall counter. After the outer loop is done, we divide the counter A by the number of masks, yielding the mean difference according to IOU for the given image.

C. Unrelated Perturbation

The final objective aims to create perturbation unrelated to the object or region of interest by maximizing the distance between perturbation and object. If we can successfully identify unrelated perturbations, this implies that it is possible to corrupt the segmentation process without directly modifying the region of the image containing the corrupted object.

Algorithm 2 shows the pseudocode for the algorithm. The inputs are a hyper parameter ϵ, the input image img, the perturbation δ, and the segmentation function f. First, the algorithm computes a matrix D with all entries $D[i, j]$ representing the minimum distance between pixel $[i, j]$ from the center of an image mask in the output $f(img+\delta)$. Since image masks can come in many different shapes, and YOLO and DETR both provide bounding boxes for objects, we take the center of the bounding box as the reference point. Next, we scan through each pixel in the image, assigning a negative value in the matrix for each corresponding pixel that falls within an image mask. The ϵ value acts as a buffer for the image masks. ϵ is a hyper parameter in this algorithm we use to specify a buffer around the image mask in which perturbations are not selected. The goal of image masks is to define the object boundary as precisely as possible, whereas bounding boxes inherently contain a buffer between the boundary and the actual object. We select a value of 1 empirically: most of the image masks we inspected require a buffer of 1 pixel. However, in practice we could perform a grid-search for the ideal value of ϵ. This allows us to ignore the pixels near the image mask. Next we multiply each value $D[i, j]$ with the maximum intensity of the pixel at entry $[i, j]$. The maximum intensity is simply the maximum perturbation value δ out of the three RGB channels of the pixel. This allows us to favor noise in regions further from segmented objects. The algorithm returns the sum of all the weighted distances divided by the number of unperturbed pixels. This provides a single value to represent the distance from all objects of the noise added.

Algorithm 2 Maximize distance between object and perturbation

$D \leftarrow 0^{L*W}$
while *I in range L, J in range W* **do**
  $D[i, j] \leftarrow \sqrt{length^2 + width^2}$
  while *B is a valid bounding box* **do**
    $D[i, J] \leftarrow min(D[i, j], \sqrt{(x - i)^2 + (y - j)^2})$
  end
end

$neg_{avg} \leftarrow (-1)\frac{\sum_{i,j} D[i,j]}{L*W}$
while *i in range L and j in range W* **do**
  while *B is a valid bounding box* **do**
    if
      i $\in$ *[x-*$\frac{l}{2} + \epsilon, x + \frac{l}{2} + \epsilon$*] and j*$\in$ *[y-*$\frac{w}{2} + \epsilon, y + \frac{w}{2} + \epsilon$*]*
    then
    end
    $D[i, j] \leftarrow -neg_{avg}$
  end
end

$\delta^{max}_{abs} \leftarrow 0^{L*w}$
while *i in range length and j in range width* **do**
  $\delta^{max}_{abs} \leftarrow max(\delta[i, j, 1], \delta[i, j, 2], \delta[i, j, 3])$
  $D[i, j] \leftarrow \delta^{max}_{abs}[i, j] * D[i, j]$
end

$unperturbed\_pixel\_count \leftarrow \sum_{(i,j),suchthat\delta^{max}_{abs}[i,j]\neq 0} 1$

return $\frac{\sum_{i,j} D[i,j]}{unperturbed\_pixel\_count}$

IV. Generating Perturbations

To complete the multi-objective optimization search, we use two different solvers to broaden the search for optimal perturbations: NSGA-II [12] and AGE-MOEA [13]. NSGA-II utilizes *Pareto rank* and *crowding distance* which help the algorithm undergo Pareto sorting. AGE-MOEA is similar to NSGA-II, with a different crowding distance formula.

For the noise applied to images, we use Gaussian noise and apply a Gaussian filter mask to each RGB value in each pixel. The filter mask used to perturb the images is represented with a direct (or explicit) encoding. For the population size of the genetic algorithm, we use 25 to limitations on computational power and the complexity of our functions. We observe empirically that 25 provides just enough of a population size to encapsulate a good solution space while also keeping a diverse solution set. In the package `pymoo` [14], we use the default parameters for the crossover. One point crossover is chosen on the pixel array and applied with a probability value $P_c = 0.5$. Then offspring filters are created by random pixel indexes. In regards to the mutation aspects, we treat pixels as the genes and perform operations similar to [3], with a probability $P_m = 0.5$, and a mutation window size of 0.01.

While we choose NSGA-II because of the diversified so-

lutions it discovers, it has been noted that it does not perform well on complex problems, and its optimization ability tends to settle on local optima rather than global optima [15]. It has been shown that different algorithms perform worse on different geometries. These geometries could be euclidean but could also be spherical or a hyperbolic hyper surface. AGE-MOEA utilizes a non-euclidean geometric approach [13]. It has been shown that the AGE-MOEA algorithm creates a more diverse solution space, because to be an ideal solution in this algorithm means to contribute to the diversity and proximity of the non-dominated front with regards to the estimated geometry. Since this is also an evolutionary algorithm, we decided to keep the parameters (mutation rate, crossover probability, etc.) and encoding the same for both solvers.

Figure 3 shows the results for the NSGA-II optimization search for YOLOv8. The goal of this search is to minimize the first two functions (f_1) and (f_2) and maximize (f_3) We observe that the search is successful: we can find perturbations that achieve minimum value (0) for $f1$ and similar for $f2$ and $f3$, meaning they affect the segmentation output. The same observations hold for DETR (Figure 4).

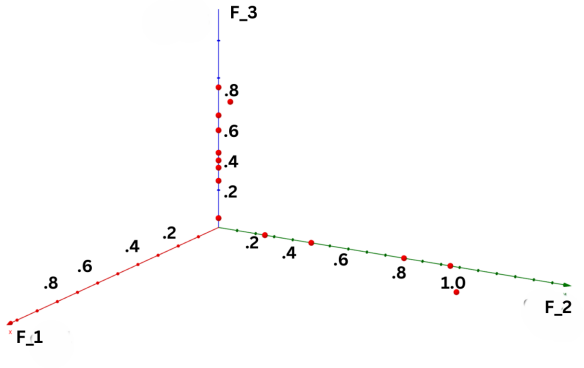

Fig. 3. NSGA-II optimization search for YOLOv8. $f1$ goal is to minimize the perturbation, $f2$ goal is to maximize the output error, and $f3$ goal is to maximize the distance between the object and perturbation.

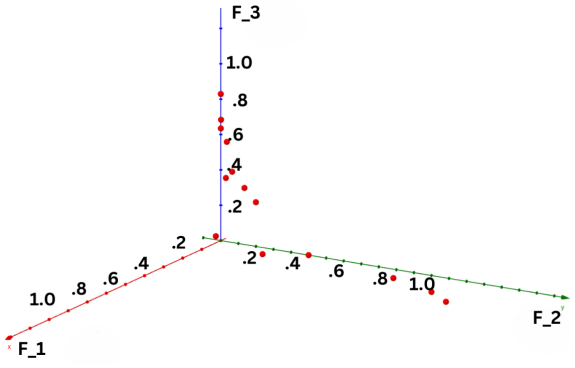

Fig. 4. NSGA-II optimization search for DETR. $f1$ goal is to minimize the perturbation, $f2$ goal is to maximize the output error, and $f3$ goal is to maximize the distance between the object and perturbation.

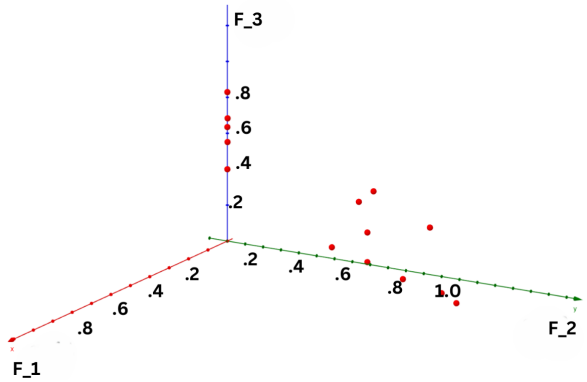

Fig. 5. AGE-MOEA optimization search for YOLOv8.

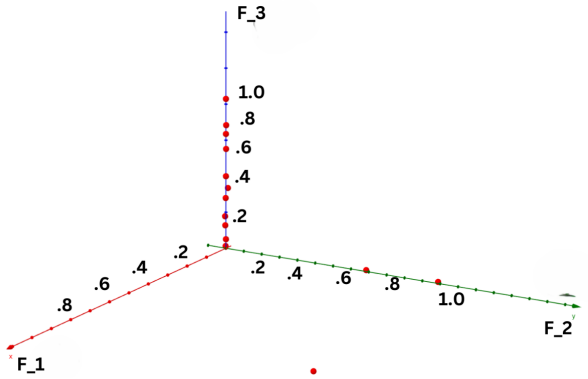

Fig. 6. AGE-MOEA optimization search for DETR.

Figures 5 and 6 show the AGE-MOEA optimization search for YOLOv8 and DETR respectively. Again the search finds undetectable perturbations that affect segmentation output. We make two observations. First, the AGE-MOEA search yields more outliers than NSGA-II. This is expected, as AGE-MOEA is designed to prioritize diversity in the solution space. Second, the runtime of NSGA-II is longer as it takes O(MN^2) while AGE-MOEA runtime is O(MN), where M is the number of objectives and N is the population size. This was proved empirically as we saw that NSGA-II took about the time AGE-MOEA took squared. This is even longer than expected, as only the population size is squared in the time complexity of NSGA-II. The extended time is expected as NSGA-II is a more complex solver and our problem is very complex. We conclude that AGE-MOEA is more effective because it gives more diverse solutions (a bigger range of effective noise) and has less runtime for the same population and generation sizes.

V. Predicting Perturbations

After applying optimal perturbation to a variety of images, we notice some images are not affected by the optimal perturbation generated by the search. This leads us to ask the following question: Could we predict if the segmentation of an image would be affected by noise? We approach this problem from a data-driven point of view. We create a

Fig. 7. Sample training image from KITTI dataset.

convolutional neural network (CNN) and train it on images from different KITTI datasets (see Figure 7 for example), and attempt to predict whether the segmenter's output for a certain image could be corrupted by noise, e.g., generated by our optimal search. Rather than generate optimal noise for each image in the dataset, which is impractical, we create an optimal noise *range*, which we find empirically to be effective across a large variety of images.

A. Definition of the Problem

We define a standard binary classifier CNN that will, given an image of any size R^{L*W*3}, output whether or not the image will be corrupted by noise. To train the predictor, we apply noise from our predefined range to a training image, and run it through the target segmenter to test whether the image will be corrupted. The noised applied is randomly chosen from the range, which could prevent the predictor from being able to generalize for a variety of perturbations on the same input image. To mitigate this over-specialization of the predictor CNN, we defined the noise range as the union of all the solutions from both optimization algorithm searches. We train three different predictor networks: (1) with images from both segmenters, YOLO+DETR; (2) with images only from YOLOv8 segementer, labeled YOLO-only; (3) with images only from DETR segementer, DETR-only.

VI. Evaluation

A. Experimental Setup

Figure 8 shows the network architecture for the predictor CNN. We implement our neural network in TensorFlow and create a standard convolution neural network. The input is an image with perturbation applied that will be automatically resized when run through the network. We start with a standard convolution layer followed by a max pooling layer and we repeat this two more times. Then we follow a flatten layer and two dense layers. The activation function is RELU, and the number of filters per convolution layer is 16. Following each convolution layer is a max pooling layer to condense our input shape into half of what the original image was. The last three layers ensure that we flatten our final layer to a single dimension, passing it to a dense layer for a prediction. We train this model on 20 epochs.

B. Results

We train the predictor networks on around 1500 images in total from various datasets from KITTI. The test dataset contains roughly 300 images. 100% accuracy was achieved

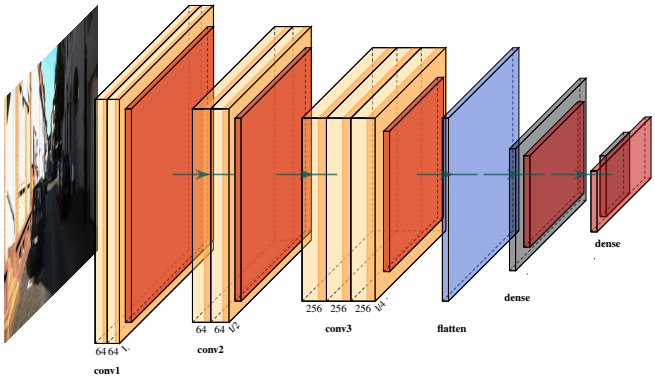

Fig. 8. Predictor CNN architecture

on our test dataset for YOLO+DETR. We select the test set to be as representative of the training data as possible. Table I shows the performance of each predictor CNN in terms of accuracy, precision, and recall achieved for the test set. The precision score is measured by $\frac{TP}{TP+FP}$, where TP are true positives and FP are false positives. Recall is measured as $\frac{TP}{TP+FN}$.

We make the following observations. First, the YOLO+DETR CNN achieves 95% recall with 100% precision, telling us that we can identify almost all images with corrupted segmentation, without any false positives. This is significant, as it means that we can detect corruption due to noise, malicious or otherwise, without access to the ground truth. Second, the YOLO-only CNN achieves 90% precision, while the DETR-only CNN achieved 94.7% precision, and both achieve 100% recall. This tells us that DETR is more susceptible to corruption due to noise than YOLO. Furthermore, the noise that YOLO is susceptible to is a subset of the noise DETR is susceptible to, allowing the DETR-only CNN to generalize to YOLO segmentation.

The above evaluations are performed with similar training and test images. We now evaluate the CNNs' ability to generalize to diverse environments using different KITTI datasets. The first new dataset consists of dark images – images with shadows overall and no direct sunlight (example in Figure 9). All models perform poorly on these images and are not able to generalize. The second new dataset consists of images with more even lighting, but in a different environment. The models are able to generalize well to this environment, achieving 80% accuracy, 76% accuracy, and 60% precision. The third new dataset contains bright images from various camera angles, e.g., birds-eye view. The models are unable to generalize for different camera angles,

TABLE I
Predictor CNN performance

Network	Precision	Recall	Accuracy
YOLO+DETR	100%	95.23%	98.4375%
YOLO-only	90%	100%	96.875 %
DETR-only	94.736844%	100%	96.875%

Fig. 9. False positive: the predictor CNN identified the image as corrupted by noise, when in reality it was not. The contributing factor to the incorrect prediction was the sunlight present in the image.

achieving 53% precision, 31% recall and 51% accuracy.

VII. Conclusion and Future Outlook

In this work, we measured the susceptibility of popular object segmentation neural networks YOLOv8 and DETR to input image perturbations. We observed that it is possible to affect the segmentation output of an object in an image by inserting noise undetectable by the human eye into the input image, in regions of the image far from the target object. We also designed and trained a convolutional neural network to detect whether an image will yield corrupt outputs, i.e., incorrect segmentation results. This effectively reverse-engineers the perturbation process, detecting problematic perturbation in input images before they are even run through the segmentation network. We observed that it is possible to predict the corruption, but the CNN predictor does not generalize well to diverse inputs, e.g., different lighting conditions or unknown environments. We believe the predictor CNN is highly dependent on both hyper-parameters and training data, meaning generalization can be improved empirically through training data and parameter tuning. The prediction method can be effective for consistent autonomy environments, e.g., a pick and place robot operating in a small warehouse region, or an autonomous train traveling the same tracks. A binary classifier for perturbation detection could be executed in parallel alongside early stages of a standard computer vision pipeline, e.g., object detection, in order to determine confidence or validity of outputs of downstream stages.

One interesting alternative to a CNN classifier that could address the shortcomings due to limited training data is a support vector machine approach. We did attempt to create a support vector machine and we found that the support vector machine achieved a 68% accuracy on the test dataset. We assume this low value is due to SVMs inability to scale well when the ratio of features per data point is very large. We chose to use a CNN because of the complexity of the problem: the environmental conditions of the images can vary highly, and CNNs are superior in complex pattern recognition compared to vector machines. However, the CNN proved to be sensitive to environmental variability as well, so a vector machine may prove superior for constrained environments with limited training data. We plan to explore this in future work.

References

[1] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020. [Online]. Available: https://arxiv.org/abs/2005.12872

[2] D. Reis, J. Kupec, J. Hong, and A. Daoudi, "Real-time flying object detection with yolov8," 2024. [Online]. Available: https://arxiv.org/abs/2305.09972

[3] N. A. V. Doan, A. Yüksel, and C.-H. Cheng, "Butterfly effect attack: Tiny and seemingly unrelated perturbations for object detection," in *2023 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2023, pp. 1–6.

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013. [Online]. Available: https://doi.org/10.1177/0278364913491297

[5] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," 2019. [Online]. Available: https://arxiv.org/abs/1903.12261

[6] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 132–142. [Online]. Available: https://doi.org/10.1145/3238147.3238187

[7] S. Wang and Z. Su, "Metamorphic testing for object detection systems," 2019. [Online]. Available: https://arxiv.org/abs/1912.12162

[8] A. N. Bhagoji, W. He, B. Li, and D. Song, "Exploring the space of black-box attacks on deep neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1712.09491

[9] M. Alzantot, Y. Sharma, S. Chakraborty, H. Zhang, C.-J. Hsieh, and M. Srivastava, "Genattack: Practical black-box attacks with gradient-free optimization," 2019. [Online]. Available: https://arxiv.org/abs/1805.11090

[10] C. Szegedy, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[11] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 39–57.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[13] A. Panichella, "An adaptive evolutionary algorithm based on non-euclidean geometry for many-objective optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 595–603. [Online]. Available: https://doi.org/10.1145/3321707.3321839

[14] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[15] J. Hao, X. Yang, C. Wang, R. Tu, and T. Zhang, "An improved nsga-ii algorithm based on adaptive weighting and searching strategy," *Applied Sciences*, vol. 12, no. 22, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/22/11573