Adversarially Robust Edge-Based Object Detection for Assuredly Autonomous Systems

Robert Canady, Xingyu Zhou, Yogesh Barve, Daniel Balasubramanian, Aniruddha Gokhale
Dept of Computer Science, Vanderbilt University, Nashville, TN
(robert.e.canady,xingyu.zhou,yogesh.d.barve,daniel.a.balasubramanian,a.gokhale)@vanderbilt.edu

***Abstract*—Edge-based and autonomous, deep learning computer vision applications, such as those used in surveillance or traffic management, must be assuredly correct and performant. However, realizing these applications in practice incurs a number of challenges. First, the constraints on edge resources precludes the use of large-sized, deep learning computer vision models. Second, the heterogeneity in edge resource types causes different execution speeds and energy consumption during model inference. Third, deep learning models are known to be vulnerable to adversarial perturbations, which can make them ineffective or lead to incorrect inferences. Although some research that addresses the first two challenges exists, defending against adversarial attacks at the edge remains mostly an unresolved problem. To that end, this paper presents techniques to realize robust and edge-based deep learning computer vision applications thereby providing a level of assured autonomy. We utilize state-of-the-art (SOTA) object detection attacks from the TOG (adversarial objectness gradient attacks) suite to design a generalized adversarial robustness evaluation procedure. It enables fast robustness evaluations on popular object detection architectures of YOLOv3, YOLOv3-tiny, and Faster R-CNN with different image classification backbones to test the robustness of these object detection models. We explore two variations of adversarial training. The first variant augments the training data with multiple types of attacks. The second variant exchanges a clean image in the training set for a randomly chosen adversarial image. Our solutions are then evaluated using the PASCAL VOC dataset. Using the first variant, we are able to improve the robustness of YOLOv3-tiny models by 1-2% mean average precision (mAP) and YOLOv3 realized an improvement of up to 17% mAP on attacked data. The second variant saw even better results in some cases with improvements in robustness of over 25% for YOLOv3. The Faster RCNN models also saw improvement, however, less substantially at around 10-15%. Yet, their mAP was improved on clean data as well.**

***Index Terms*—machine learning, edge computing, adversarial robustness, object detection**

I. Introduction

Computer vision is a field that has applications in many areas including surveillance, medicine, traffic management, autonomous vehicles, manufacturing, and public safety. Reliable, accurate and real-time object detection and identification are crucial in order for these applications to function correctly. For instance, a smart home security camera responsible for detecting potential intruders must reliably detect and identify people and suspicious behavior in its field of vision.

Computer vision applications are both data-intensive (e.g., involving continuous stream of images) and compute-intensive (e.g., object detection and classification computations on these stream of images). Although the cloud provides a resource-rich environment to handle the computational needs of such applications, the stringent response time requirements of these applications (e.g., autonomous vehicles, navigational guidance to the visually impaired) requires that the computations on the critical path of these applications be deployed at the edge while utilizing the cloud for longer time-scale computations, such as aggregation and other forms of batch processing.

Realizing reliable and robust computer vision applications at the computing edge is, however, fraught with many challenges. First, many of the inferencing tasks in such computer vision applications rely on deep machine learning models. For instance, object detection models based on convolutional neural network (CNN) architectures like YOLO [1] are often used in computer vision tasks due to their inference speed and accuracy. However, these deep learning models are very large and computationally intensive both for training and inferencing. The constraints on the resources at the edge therefore preclude the direct use of such models. Hence, recent research has focused on techniques to define smaller versions of these models to suit the edge devices, e.g., YOLO-Lite [2].

Second, although miniature versions of these deep learned models exist, the computing edge illustrates significant heterogeneity in the types of available resources including a range of hardware accelerator devices. For instance, devices like many generations of Raspberry Pis and Beagle Bones along with hardware accelerators, such as the Jetson Nano GPU, Neural compute sticks like CORAL USB, and FPGAs like Ultra96v2, are now commonplace. Although these devices are able to execute the miniature versions of the deep learned models, each device type demonstrates a different execution and energy consumption profile as revealed by our prior preliminary work [3]. Each device might also rely on a different deep learning framework, which adds another layer of complexity.

Third, the problem with deep learning, especially as used in computer vision, is the general vulnerability of these models to adversarial machine learning (AML) attacks [4]. In these attacks, often times one cannot tell that the image has been changed, but only a slight perturbation that has been added causes the deep learning model to make wrong predictions. Although many of these attacks have been carried out on image classification tasks, attacks on object detectors are also receiving more attention lately, particularly with increasing number of applications that utilize computer vision models, e.g., in autonomous driving [5]. There have been several documented attacks, such as ShapeShifter [6], Poltergeist [7], DAG [8], Imperceptible Background Patches [9] and TOG [10], [11] to name a few that have been very successful at causing anything from vanishing labels to fabrication of labels.

Defending against AML attacks is a difficult problem; it becomes even more difficult when solutions must be devised to work effectively within the stringent constraints of edge devices while ensuring availability, reliability, and timeliness of the edge-based computer vision applications. This paper addresses this urgent need and investigates defense and deployment approaches towards the robustness of these smaller, edge-based models against adversarial machine learning attacks. To that end, we make the following contributions:

1) Design and demonstrate a generalized adversarial robustness evaluation framework for edge-based object detection models from both a software algorithm and hardware capability perspective;
2) Propose two new variations of data augmentation-based adversarial training methods which we call Quartet Adversarial Training (QUAT (1.0) / QUAT (2.0)) that not only create more robust edge-based models for object detection, but can also be used to create more robust cloud-based models;
3) Define an effective deployment approach to realize robust edge-based computer vision applications; and
4) Validate the efficacy of the proposed QUAT algorithms using the Pascal VOC dataset [12].

The rest of the paper is organized as follows. Section II provides background and related research comparing it to our work; Section III presents the attack workflow and description of our defense mechanism; Section IV presents the experimental testbed and evaluates the defense against multiple different attacks; and finally Section V provides concluding remarks and discusses future directions and threats to validity.

II. Background and Related Work

To make this paper self-contained, we first provide background information on the use of deep learning in computer vision focusing on object detection and adversarial machine learning. We then describe related prior efforts comparing them to our proposed ideas and highlight our contributions.

A. Overview of Object Detection in Computer Vision

Object detection in computer vision can be broken into two categories: Two-stage and Single Shot, each with its advantages and disadvantages. Two-stage object detectors like Faster RCNN [13] have a region proposal network in the first stage that narrows down the number of Regions of Interests (ROIs) to approximately 2K. In the second stage, classification is carried out on each ROI by first being fed through a CNN and then to fully connected layers to complete the classification and refine the bounding box. Sending each detected ROI through a CNN, however, slows down training and inferencing. Although two-stage detectors are accurate, they are still considered slower.

YOLO (You Only Look Once) [1] is a famous single shot object detection technique with numerous versions being published over the years: YOLOv1 [1] originally and most recently YOLOv3 [14] and YOLOv4 [15]. In this paper, we have utilized YOLOv3 as it is extremely fast and accurate. Moreover, there is ample support and documentation available. YOLOv3 uses DarkNet-53 with 53 convolutional layers as its feature extractor backbone. YOLOv3 uses a Feature Pyramid Network to detect objects that are near or far away. YOLOv3-tiny is a smaller version of YOLOv3 with fewer weights but this model trades off accuracy for speed.

B. Prior Work on Edge-based Computer Vision Applications

There have been several recent works, such as OpenDataCam [16] and Coral-Pie [17] that demonstrate different object detection applications at the edge and the need for edge accelerators. In Coral-Pie, the vehicle tracking application uses two Raspberry Pis connected to a Coral USB. The authors did not use YOLOv3 for object detection because it was too computationally expensive for the CORAL USB. OpenDataCam is an open source tool for monitoring and tracking moving objects in a live video stream. This application uses YOLOv3 on a desktop machine and recommends using YOLOv3-tiny for edge devices like Jetson Nano.

C. Prior Work on Adversarial Machine Learning Attacks and Defenses

Adversarial machine learning (AML) attacks on deep learning is a relatively new field with its start in classic linear regression models [18]. The work on adversarial evasion attacks [19] led to the seminal work on adversarial attacks on deep learning models [20]. It attracted much attention in recent years given the rapid development in deep learning especially in computer vision tasks [4], [21], [22]. Based on how much information is known to the attacker, AML comprises both white box and black box attacks. Most attacks are based on the white box setting where the model details or full training data is known. In contrast, the work in [23] implements these attacks in a black-box scenario, where the attacker only has access to the output of the model given an input. One important property of AML is that adversarial examples usually have a high transferability between models [24]. This has become an important metric for attacks and defenses, i.e., whether a defense can defend against multiple attack types or whether an attack can break multiple model types.

There has been much work on finding suitable defenses for these attacks. Defenses fall into two categories: reactive and proactive, where the former corresponds to detection of adversarial images and the latter to making the model more robust. Currently, the most successful single-model defense is the proactive method of adversarial training [25]. The idea is to augment the training data with adversarial examples so that the model will take into account the data distribution shifts caused by perturbations, and be able to correctly classify the adversarial image. There have been other defense mechanisms that utilize data augmentation [26] and/or pre-processing [27],

[28] where the idea is to remove/destroy the perturbations to mitigate the adversarial impact. The combination of these data transformations with adversarial training [29] have also been proposed as a set of defense strategies.

D. Prior Work on Object Detector Attacks and Defenses

Compared to misleading classification in image recognition tasks, object detectors combine both object localization and recognition. This leads to more flexible attack settings available in both phases. Some of the more recent attacks include DAG [8], Imperceptible Background Patches [9], ShapeShifter [6], Poltergeist [7], and TOG [10]. In this paper, we have used the TOG attack suite for our adversarial training (see Section III) as well as Poltergeist for evaluation.

Since adversarial ML attacks on object detectors is still a relatively new field, literature on defense techniques is still scarce. Efforts, such as [30], have used a two-stage adversarial training algorithm to improve the robustness in safety-critical scenarios. The authors use pre-trained models that they then fine-tune on data that they attack using a multi-step PGD method. In [31] the authors present a similar adversarial training approach with a model trained on PGD attacked data. They however only use one type of attack to augment their training data. In a recent example [32], authors update the adversarial examples dynamically during training by selecting the strongest attacked image based on the image classifier and localization branches while evolving with the detector.

E. Differentiating our Proposed Work from Prior Efforts

Our work focuses on edge-based (and thereby also cloud-based) object detection adversarial machine learning defense strategy, which to our knowledge has not been researched extensively. Since there has not been much work on the defense side of object detection, we attempt defense strategies that have worked on image classification tasks. We have devised different forms of adversarial training where we augment the training data with four types of attacks each having a different purpose or swap out the clean image in the training set for a random adversarial equivalent. Most attacks like PGD [25] or C&W [22] attempt to cause the classifier to incorrectly classify the object. In these object detection attacks, each attack attempts to affect the loss function in a different way to either cause vanishing, extra, or incorrect labels.

We evaluate the attacks and defenses on both edge-based and traditional cloud-based object detection models. We also evaluate how well these models perform on different cloud and edge hardware. Our work builds upon the ideas from Section II-D, where the attacks were devised using strategies from TOG [10]. We follow the guidelines of adversarial machine learning evaluation [33] to build a reliable attack evaluation workflow for object detectors in edge/cloud scenarios.

III. Design of Edge-based Adversarially Robust Models

This section presents our proposed QUAT defense strategy.

A. Overview of the Proposed QUAT Approach

A high-level workflow of our proposed Quartet Adversarial Training, or QUAT, approach is presented in Figure 1. The intent is to develop adversarially robust, edge-based object detection models. For this, we need a way to subject the model(s) to a range of adversarial attacks. The TOG suite [10] provides such a capability from which we derive adversarially trained images as shown in Figure 1. These then serve as the training data to define adversarially robust models suitable for deployment at the edge using two variants of the QUAT process that we have designed. These adversarially robust models are then evaluated using a set of test data and compared with regular models that are not adversarially robust.

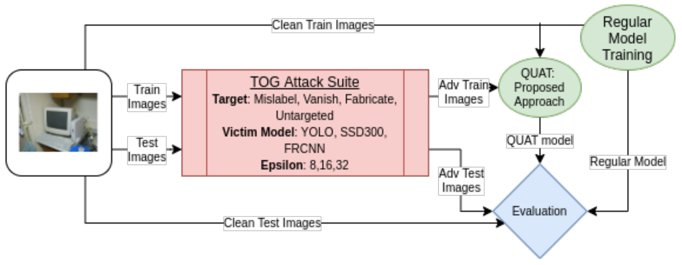


Fig. 1. Worflow for Designing Edge-Based Adversarially Robust Models

B. Attack Injection and Adversarial Training using the TOG Attack Suite

To help design a defense mechanism against adversarial attacks on machine learned models, we first needed to understand whether these attacks affect the losses differently and if so, in what way. Thus, we sought a systematic approach that could provide us a broad coverage of attacks on models that could highlight these nuances. To that end we chose TOG (Targeted Adversarial Objectness Gradient) [10], which defines a suite of attacks on object detectors. TOG provides a systematic adversarial robustness evaluation workflow for object detection. Moreover, TOG works on both the One Shot and Two Stage detectors, and further, the targeted attacks in the TOG suite impact each model a little differently so that the model under consideration can then be defended using a more diverse adversarial training set.

The TOG suite ensembles several physics-based attacks on object detectors and therefore provides a simple integrated evaluation procedure for common object detectors, i.e., different loss functions. The object detection loss can be broken down into three different losses: $\mathcal{L}_{obj}$ for detecting the existence of an object, $\mathcal{L}_{bbox}$ for the location and dimension of the bounding boxes, and $\mathcal{L}_{cls}$ for the detected object's class. Moreover, TOG can carry out four different types of attacks:

- *Untargeted (U):* Causes the model to incorrectly detect and classify an image without targeting any specific object.
- *Vanishing (V):* Causes the model to not detect any objects in the image when there are objects.

- *Mislabel (M):* Causes objects to be detected, but classified incorrectly.
- *Fabricate (F):* Causes additional false objects to be detected with high confidence.

There are several hyperparameters that can be set for the attack, such as number of iterations, confidence to filter predictions, perturbation bound ϵ, and attack learning rate α. ϵ is essentially the strength of the attack or the maximum distortion. It is usually in the range of 8/255–32/255, where 255 comes from the fact that an 8-bit pixel has 256 possible values. The confidence is in the range of 0.0-1.0, and is the threshold that is used to output the detections based on the confidence in the detection. The attack learning rate is the step size of the attack and is set at 2/255, per convention.

C. Proposed Defense Strategy: QUAT

Although TOG provides systematic evaluation of attacks on object detectors, it does not utilize the orthogonality property underlying these attacks. We exploit this orthogonality in the attack types to conduct adversarial data augmentation for object detection model training and thereby seek more adversarially robust models. Based on this idea, we then propose a defensive strategy for object detection tasks at the edge and the cloud. To minimize deployment costs, we make use of pre-trained models and defensive techniques. To realize our approach, we faced the following challenges:

Challenge 1: Although adversarial training is widely used as the single-model deterministic inference defense to AML, it causes the clean mean average precision (mAP) to drop. By clean, we refer to data that has not been perturbed whereas the attacked data is called adversarial data. While these defenses are promising, our aim is to limit the decrease in clean mAP as much as possible because we want to preserve the model's performance in normal circumstances. Since the technique is compute-intensive, much more so for the full YOLOv3 model than YOLOv3-tiny, we surmise that by combining all four strong attack types from TOG into the adversarial training – an approach we call QUAT shown in Figure 1 – will improve the robustness of these models and the clean accuracy.

Challenge 2: Machine learned models require large training datasets and compute power. Further, for adversarial training, the original model must encounter a range of perturbations/corruptions so the model can generalize better to new situations. The challenge is thus to find a diverse adversarial training dataset without inducing more training costs. While not initially important, this will be important when the models need to be retrained with new data.

To address these challenges, we present two variants of a defensive approach called QUAT, where the second variant was developed based on lessons learned from the first.

1) QUAT (1.0): In the first variant, we added the attacked images to the training set, which in turn formulates a type of data augmentation technique [34]. The detailed operation of QUAT (1.0) is described in Algorithm 1, where we split the training data into four subsets U (untargeted), V (vanishing), M (mislabeling) and F (fabrication) that are mutually exclusive and correspond to the 4 attack types in TOG, but together make up the full training set. This is shown on line 1. We then attack each mutually exclusive subset with either untargeted, mislabeling, vanishing, or fabrication attacks with ϵ values of 8/255, 16/255 and 32/255. The different $epsilon$ values results in models that are trained on different strengths of attacks thereby allowing us to see how training is impacted with different attack strengths. This operation is carried out on lines 2-11. After each image has been attacked, we save the attacked images with the ground-truth bounding box information in the traditional YOLO training format (class name, confidence, left, top, right, bottom). The attacked datasets are combined by their ϵ values so that we can train separate models on all 4 attacks with the same epsilon value. On line 14 we then combine the clean and attacked datasets for three QUAT (1.0) datasets of different attack strengths.

Algorithm 1: Generation of QUAT (1.0) Images

```
Require: $X$: original training samples; $\epsilon$: maximum perturbation bounds ($L_\infty$ constraint); $\mathcal{L}$: adversarial loss; $\mathcal{L}_{obj}$: objectness loss; $\Gamma$: sign function; $O$: auxiliary target detection; $\prod_{x,\epsilon}[*]$: is the projection onto a hypersphere with a radius $\epsilon$ centered at $x \in X$ in $L_p$ norm; $T$: attack iterations; $\alpha$: attack learning rate; $W$: model weights
1:  $V \cup M \cup F \cup U = X$ s.t. $V \cap M \cap F \cap U = \emptyset$
2:  for $\epsilon = 8, 16, 32$ do
3:      for $v \in V, m \in M, f \in F, u \in U$ do
4:          for $i = 1...T$ do
5:              $v'_{t\epsilon} = \prod_{x,\epsilon}\left[v'_{t-1} - \alpha\Gamma\left(\frac{\partial\mathcal{L}_{obj}(v'_{t-1}; \emptyset; W)}{\partial v'_{t-1}}\right)\right]$
6:              $m'_{t\epsilon} = \prod_{x,\epsilon}\left[m'_{t-1} - \alpha\Gamma\left(\frac{\partial\mathcal{L}(m'_{t-1}; O^*(x); W)}{\partial m'_{t-1}}\right)\right]$
7:              $f'_{t\epsilon} = \prod_{x,\epsilon}\left[f'_{t-1} + \alpha\Gamma\left(\frac{\partial\mathcal{L}_{obj}(f'_{t-1}; \emptyset; W)}{\partial f'_{t-1}}\right)\right]$
8:              $u'_{t\epsilon} = \prod_{x,\epsilon}\left[u'_{t-1} + \alpha\Gamma\left(\frac{\partial\mathcal{L}(u'_{t-1}; \hat{O}(x); W)}{\partial u'_{t-1}}\right)\right]$
9:          end for
10:     end for
11: end for
12: $v'_{t\epsilon} \in V'_{t\epsilon}, m'_{t\epsilon} \in M'_{t\epsilon}, f'_{t\epsilon} \in F'_{t\epsilon}, u'_{t\epsilon} \in U'_{t\epsilon}$
13: $V'_{t\epsilon} \cup M'_{t\epsilon} \cup F'_{t\epsilon} \cup U'_{t\epsilon} = X'_{t\epsilon}$
14: $Q_\epsilon = X \bigcup X'_{t\epsilon}$
```

2) QUAT (2.0): Our evaluation of QUAT (1.0) (see Section IV-C) reveals long training times due to the training set being two times the size of the regular training set and drop in clean accuracy as its key drawbacks. Hence, we propose QUAT (2.0), which is a rapid transfer-based adversarial tuning framework for generalized object detection tasks under adversarial settings. After implementing QUAT (1.0), we realized that we could give the model even more diverse data by training on multiple ϵ values as well as images created by attacking different victim models. Our reasoning was that this would also limit transferabililty of other adversarial examples as well as different attack strengths.

Our framework makes use of a pretrained object detection model and fine-tunes it with precomputed adversarial perturbations. We computed these perturbations in the same way as in Algorithm 1 except that instead of splitting the training set into four mutually exclusive subsets, we attack the entire training set with each different attack. This way, when randomly selecting the image to swap, it could be any of the attacks with any bound on any victim model. It could also just be a clean example. This transfer learning procedure induces limited compute burden while seeking a more balanced performance between natural robustness and adversarial robustness. Compared to the normal object detection model training that usually needs more than 100 iterations to reach an optimal status, our framework only induces a limited $15 - 25$ number of iterations.

As a result, we can formulate the optimization goal for the proposed method as the following:

$$\min_{adv \in \boldsymbol{TOG}} \sum L_{\text{nat}}(ObjDet) * \alpha_{nat} + L_{\text{adv}}(ObjDet) * \alpha_{adv}$$
subject to:
$$\alpha_{nat} + \alpha_{adv} = 1 \tag{1}$$

In this way, we seek an overall goal through the minimization of the empirical loss as a weighted-sum of adversarial loss and natural loss. We solve this optimization problem in an iterative way using the proposed adversarial parameter tuning procedure.

Algorithm 2: Swap of Clean Images During Data Loading

```
Require: $X$: original training samples; $\epsilon$: perturbation
    bounds (8,16,32); $M$: victim model (FRCNN, YOLO,
    SSD); $T$: type of TOG attack (fab, van, mis, un, clean);
    $A_{\epsilon MT}$: adversarial training samples; $Rand$: random
    selection function; $load$: load data for training.
 1: for $x \in X$ do
 2:     $e = Rand(\epsilon)$
 3:     $m = Rand(M)$
 4:     $t = Rand(T)$
 5:     if $t == clean$ then
 6:         $load(x)$
 7:     else
 8:         $load(a_{emt})$ where $a_{emt} \in A_{\epsilon MT}$
 9:     end if
10: end for
```

Instead of adding images to the training set, we randomly swap out a clean image for an adversarially attacked image. We hypothesized this would help the models generalize better to unseen attacks, as well as cut down the training time by cutting the adversarial dataset in half. We show the QUAT (2.0) data loading process in Algorithm 2. On lines 2,3,4 we show how for each image in the training set, it is random whether a clean or attacked image is loaded. If it is clean then, the normal image is loaded on line 6. If it is to be an adversarial image, then it is loaded based upon the victim model, attack type, and ϵ, shown on line 8.

TABLE I
DESCRIPTION OF DATASETS USED FOR EACH QUAT ITERATION (IT.) (F=FABRICATION, M=MISLABELING, U=UNTARGETED, V=VANISHING, Y=YOLOV3, FR=FASTERRCNN, S=SSD300)

QUAT It.	Dataset	Target Model	Attack	Strength	Total #
1.0, 2.0	Clean VOC Train	N/A	N/A	N/A	1
1.0	Adversarial VOC Train	Y	F,M,U,V	8,16,32	12
2.0	Adversarial VOC Train	Y,FR,S	F,M,U,V	8,16,32	36
1.0, 2.0	Clean VOC Test	N/A	N/A	N/A	1
1.0, 2.0	Adversarial VOC Test	Y,FR,S	F,M,U,V	8,16,32	36
2.0	Clean VOC Test Edge	N/A	N/A	N/A	1
2.0	Adversarial VOC Test Edge	Y,FR,S	F,M,U,V	8,16,32	36

IV. EXPERIMENTAL EVALUATION

This section evaluates the efficacy of our proposed QUAT approach validating the claims we made.

A. Object Detection Datasets and Models Used in Evaluation

We have used the PASCAL Visual Object Classes (VOC) [12] benchmark for object detection in the evaluations. The VOC 2007 train/validation/test set contains 9,963 images with 26,640 annotated objects. We also used the VOC 2012 data for which the train/validation data has 11,530 images with 27,450 region of interest (ROI) annotated objects. The VOC 2007 is split around 50% and then combined with the 2012 train/validation data for the full training set per convention.

To simplify the experimentation, for QUAT (1.0), we used object detectors YOLOv3 and YOLOv3-tiny because these models are known for their inference speed as well as high mAP. For QUAT (2.0), we used YOLOv3, YOLOv3-tiny, and also Faster R-CNN with either ResNet50, MobileNet v3-large or MobileNet v3-large-320 backbones because of the ease of deployment of these models as well as their performance improvements over older methods. The ResNet50 model is used as the cloud-based model while the MobileNet models constitute the edge-based models. Tensorflow is used as the Deep Learning framework for the attacks, Darknet is used to train the YOLOv3 models, and Pytorch is used for training the Faster R-CNN models.

B. General Evaluation Methodology

To determine how well the object detection models perform on datasets like PASCAL-VOC, they are judged on their inference time and their mean average precision (mAP). The mAP is calculated using a metric called *Intersection over Union (IoU).* The higher the mAP the better, but its semantics for object detection are different compared to image classification accuracy, where the classification is either correct or incorrect. In contrast, the goal of object detection is to draw bounding boxes around objects and then correctly classify the object(s). To calculate the mAP, the analyst needs the ground-truth and predicted bounding box coordinates, which can then be used to calculate the IoU. The IoU is calculated as the amount the predicted bounding box overlaps with the ground-truth bounding box divided by the total area of the union of both boxes.

To determine the efficacy of the model, the analyst sets a threshold percentage for the overlap. The threshold is usually set at 0.5 per convention and because of the fact that humans can barely tell the difference between 0.3 and 0.5 IoU. For some different datasets or competitions, a different confidence threshold is used. The mAP is then calculated by drawing precision-recall curves with the IoU set at different thresholds. This is done for each class, and at this point it is just the average precision (AP). The average AP across all classes is then the mAP. The hypothesis is that the mAP value of the adversarially robust model will be larger than the mAP value of the original, adversarially vulnerable model.

To that end, we trained all YOLOv3 models from scratch, both for QUAT and non adversarial, on the PASCAL-VOC dataset. We utilized networks that had been pre-trained on ImageNet and COCO for the Faster R-CNN QUAT (2.0) models. All of the training was conducted on a desktop (Server) with a 12 Core/24 Thread AMD Ryzen 9 3900x processor, 32 GB RAM, an 8 GB NVIDIA RTX 2060 Super GPU, and 2TB of SSD. The testing was carried out on the server (representing a cloud server) as well as a Jetson TX2 device (representing the edge device).

To evaluate how well these models, i.e., QUAT (1.0)/(2.0) versus regular, performed, we calculated the average precision (AP) and mean Average Precision (mAP) at 0.5 confidence (per convention), and recorded the inference time.

C. QUAT (1.0) Evaluation

The combined PASCAL VOC train/validation dataset has 16,551 images and their associated annotated bounding boxes. The adversarial datasets for QUAT (1.0) vary in the number of images because only certain successful attacked images were saved. The adversarial PASCAL VOC train/val dataset then has a total of either 27,232, 26,592, or 26,312 images. The random sampling in this hybrid dataset manipulates the ratio of natural and adversarial loss in the overall empirical loss minimization optimization.

We trained each tiny model for 100 and 200 iterations, and each full model for 50,000 iterations using the DarkNet framework including the adversarial images into the training data. We chose 100, 200 and 50,000 iterations, respectively, because we wanted to train all of our models for the same number of iterations to get a baseline. We also noticed that the model was trained well enough after this point to gain valuable insights.

We ran all tests with a detection confidence of 0.5, and set the input resolution of the network to 416 pixels for both YOLOv3 and YOLOv3-tiny. These are each standard values to use for the PASCAL VOC dataset. There are different numbers that can be used for the input resolution. We chose to go with a lower number because with an edge scenario we wanted the model to predict faster and be in a more constrained scenario.

1) Processing Time: The YOLOv3-tiny models processed the images faster, which was to be expected. It is apparent that the attack strength and time to process images are related. The stronger attacks (i.e. $\epsilon = 32/255$) resulted in a shorter processing time, while the clean data took the longest to process. This may be due to the fact that the models found less objects in the attacked images which could lead to shorter processing time. The YOLOv3-tiny QUAT (1.0) models were able to process the images faster on average than the regularly trained model. We noticed the opposite to occur for the YOLOv3 models. We discuss this along with other resource-aware defense approaches in Section V.

2) mAP on Clean Data: We refer to clean data as the non perturbed or non attacked data. The models we trained did not attain state of the art (SOTA) mAP. This is due to the fact that we wanted to train each model from scratch to get a baseline robustness, and to be able to compare the effects of QUAT versus traditional training. Although they are not reporting the highest mAP, we believe using these to be a better comparison than using SOTA pre-trained models against the QUAT models. In the future, with the knowledge of whether QUAT is effective, we will try to fine-tune the models to achieve SOTA results.

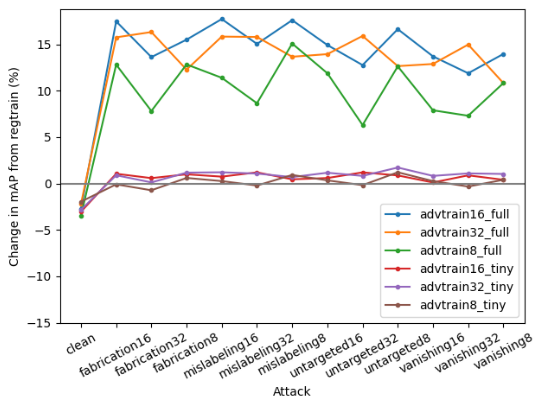


Fig. 2. Comparison of each QUAT(1.0) model to the regularly trained model $[mAP(QUAT) - mAP(REG)]$. These results demonstrate the improvement QUAT(1.0) had over the regularly trained models on attacked data

3) mAP on Adversarial Data: We now discuss the results of evaluating QUAT (1.0) models on the attacked data. We include a figure (see. Figure 2) that shows how each QUAT (1.0) model compared to the regularly trained model for each attack. If there is a negative value, then the regular model has a higher mAP and if it is a positive value, the QUAT (1.0) model has a higher mAP. All of the attacks are very effective against the YOLOv3-tiny models in terms of decreasing their mAP. The regtrain_tiny model has the highest clean mAP but then records the lowest mAP for each attack. While the regularly trained model did achieve higher mAP on the clean data by 1-3%, the QUAT (1.0) models achieved higher mAP up to 17%, on each attacked dataset.

The ϵ of the attack appears to have affected the regular models more than the QUAT (1.0) models meaning that for ϵ of 32 the AP decreased more for the regularly trained model than for the QUAT (1.0) models. This was the same pattern for each attack. It should be noted that the ϵ=32/255 attacks were the most effective attacks against each model. We also found that the tiny and full models trained on data that had been perturbed with ϵ=16/255 performed the best across the models.

The QUAT (1.0) YOLOv3 models show that the technique does improve robustness as seen in Figure 2. It can also be seen in the YOLOv3-tiny models as well, but by not as wide of a margin.

4) Transferability of Defense: To study the transferability of the defense to other attacks, we evaluated the models against attacks designed for SSD-300, SSD-512 and Faster R-CNN. For these attacks we only use ϵ=8/255. As can be seen in Figure 3, the attacks had roughly the same effect on the YOLOv3-tiny models as the attacks tailored specifically for YOLO. The transfer attacks are however not as successful on the regularly trained Full YOLO model as well as the QUAT (1.0) YOLO models. The QUAT (1.0) models outperformed the regularly trained model against these transfer attacks. This shows that the QUAT (1.0) defense does transfer to other attacks targeting different victim models.

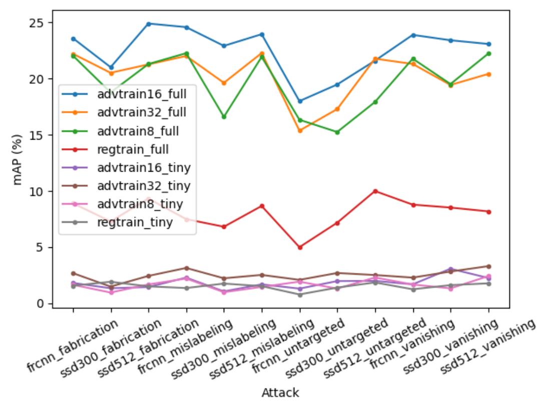


Fig. 3. Transfer Attack Results for both YOLOv3 and YOLOv3-tiny

D. *QUAT (2.0) Evaluation*

Now we will discuss how QUAT(2.0) performed against clean and adversarial data in terms of mAP as well as inference time. The models were evaluated on the Server as well as the Jetson TX2.

1) Processing Time: As can be seen in Figures 4 and 5, the Cloud-based and Edge-based YOLOv3 were faster than their equivalent Faster RCNN models. This was to be expected due to the fact that Faster RCNN is a two stage object detector compared to YOLOv3 being a one shot detector. Comparing each Faster RCNN to their QUAT (2.0) counterpart, we saw the Faster RCNN models oscillate in terms of faster processing speed depending on the attack. We saw something different with YOLOv3 models, meaning the regularly trained models processed their detections more quickly than the QUAT (2.0) model.

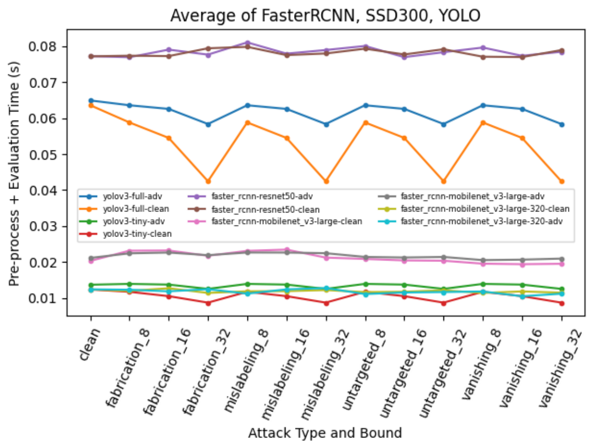


Fig. 4. Averaged Evaluation Time of Server and Edge-Based Models on Server for QUAT (2.0). This is here to help understand the processing speed and potential trade-offs of using Faster RCNN or YOLOv3.

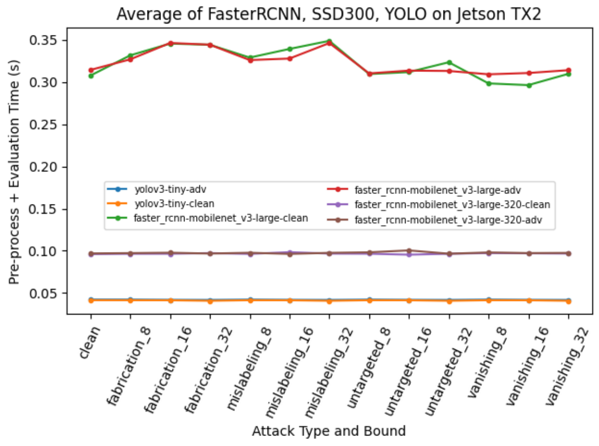


Fig. 5. Averaged Evaluation Time of Edge-Based Models on Edge Device for QUAT (2.0). This is here to help understand the processing speed and potential trade-offs of using Faster RCNN or YOLOv3.

Another point that should be noted is the increase in processing time on the Jetson TX2 which can be seen in Figures 4 and 5. While it is to be expected that the Jetson TX2 would take longer during inference, it is important to know how much and directly to compare it to the same model on a Server-type machine. Some of the models on the Server were able to evaluate images in between $10-15ms$ and $75-80ms$ whereas the same models on the Edge Device were only able to evaluate images between $50ms$ and $350ms$.

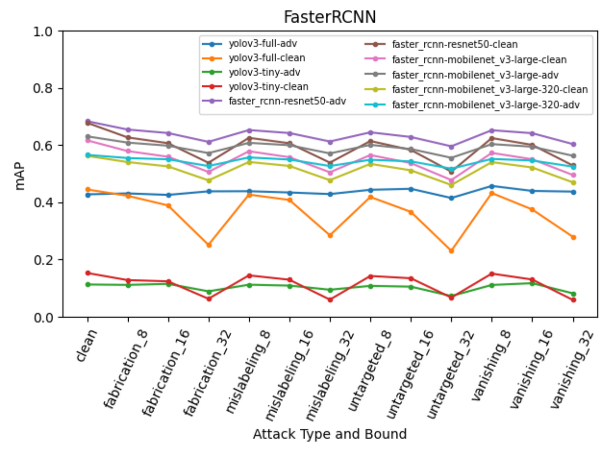
Another thing to note is that for the YOLOv3 and somewhat for the YOLOV3-tiny models, their inference time decreased as the attack strength increased. We hypothesize this is due to the fact that the model was making less predictions than the QUAT (2.0) models which stayed fairly consistent.

2) mAP on Clean Data: As can be seen in Figures 6 and 7, the QUAT (2.0) YOLOV3 models slightly underperformed compared to their regularly trained counterparts. The QUAT (2.0) Faster RCNN models actually performed the same, or better than their clean counterpart on clean data. This is not usually the case, and was not even the case for QUAT (1.0). One hypothesis could be that the QUAT(1.0) models overfitted to the adversarial training data because it was twice the size as the original training data. This has been one of the major downfalls of Adversarial Training and a very positive result for this research area. This shows that we may not always have to sacrifice clean accuracy for robustness, which is an important step in deploying deep learning applications in safety-assured situations.

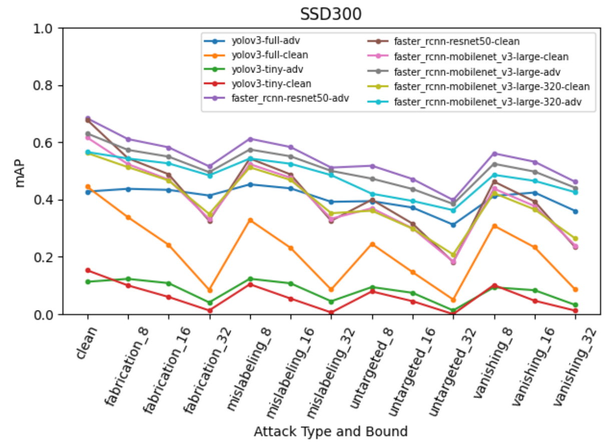
3) mAP for Adversarial Data: The QUAT (2.0) models also performed better than their clean counterpart on adversarial data (see Figures 6 and 7). It is interesting to look at how the models performed on adversarial data generated for different victim models. It appears that the SSD300 attacks were strongest across each model, in particular, the Untargeted attack with $\epsilon = 32$. Also, a closer look at Figures 6 and 7 reveal that the ϵ of the attack affected the models trained on clean data more than the QUAT (2.0) models. There is a much steeper drop off in adversarial accuracy when going from 8-16 and then from 16-32. This shows that the QUAT (2.0) models were able to generalize better to the attacks overall but also to attacks of different perturbation bounds. Seen in Figure 7, the models evaluated on the Jetson TX2 mirror the results of the models ran on the Server in terms of mAP. As discussed earlier, there were differences in their inference time.

4) mAP on Poltergeist Adversarial Data: The last evaluation we did of our QUAT (2.0) approach was on a purely black-box as well as physical attack called Poltergeist [7]. A physical attack is carried out on the sensor or detected objects instead of attacking through software. This particular attack causes misclassification by using acoustic signals directed at the inertial sensors. The authors of Poltergeist used it in an autonomous driving scenario, but for this paper it is a good example of a natural style corruption because it is representative of a camera shaking due to wind or being out of focus. There are three values that can be changed to cause different variations of the corruption. They help determine whether the corruption will be a radial, linear, or rotational motion blur as well as the extent to which the image is blurred.

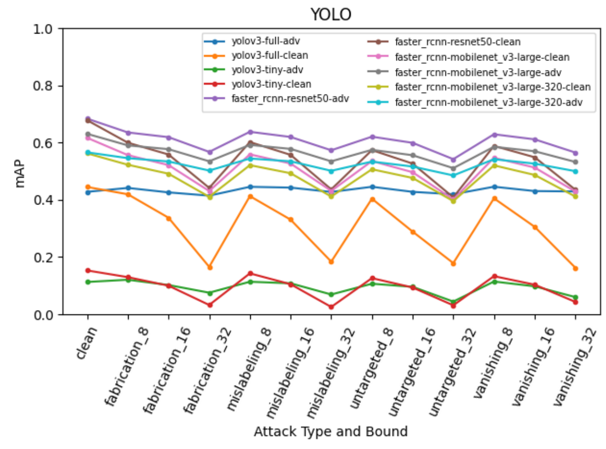
The attack was able to drop the QUAT(2.0) and regular YOLOV3-tiny models to 0% mAP across all variations of corruption. The YOLOV3 models were still able to detect some objects correctly, but their mAP did decrease. The highest mAP that QUAT (2.0) YOLOV3 was able to reach was 28.6% while the best mAP for regular YOLOV3 was only 14.3%. The QUAT (2.0) Faster RCNN models with ResNet50, Mobilenet-v3-large, Mobilenet-v3-large-320 were able to achieve on average 7.7%, 6.2%, and 8.1% mAP, respectively, while their regular counterparts were able to achieve 7.4%, 5.7%, and 6.0% mAP, respectively. The QUAT (2.0) models saw a less substantial performance gain than in the previous sections, but still outperformed their regular counterparts.



(a)

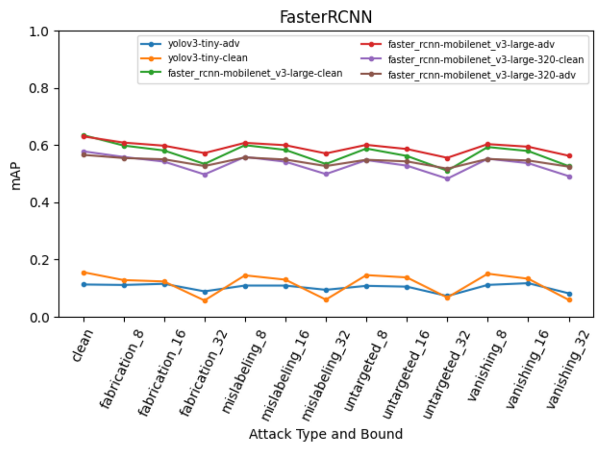


(b)

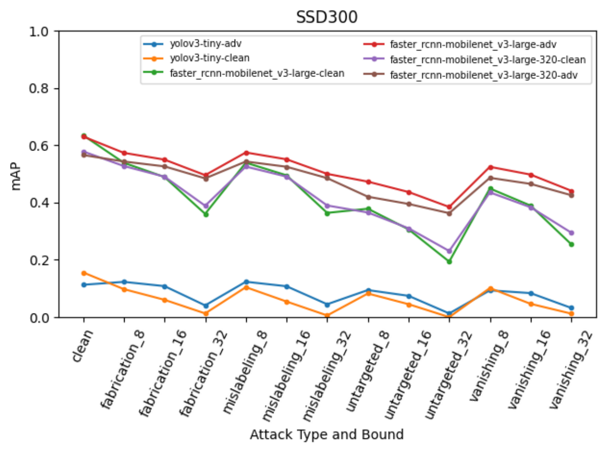


(c)

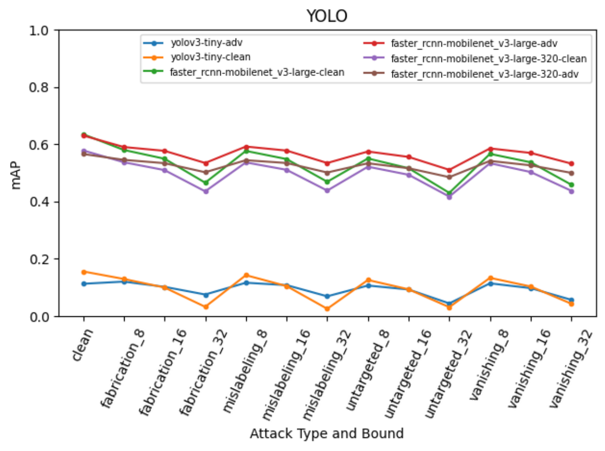
Fig. 6. Evaluation of Server and Edge-Based Models on Server Using Data Attacked with (a) Faster RCNN (b) SSD300 and (c) YOLO Architecture. This shows the efficacy of the proposed QUAT(2.0) algorithm.



(a)



(b)



(c)

Fig. 7. Evaluation of Edge-Based Models on Edge Device Using Data Attacked with (a) Faster RCNN (b) SSD300 and (c) YOLO Architecture. This shows the efficacy of the proposed QUAT(2.0) algorithm.

E. Further discussion on mAP improvement in Comparison to other Defense Methods

In a related defense work [32] for object detection, the authors report mAP improvements of 1.1% mAP on the COCO dataset. While this is not directly comparable because we are looking at the Pascal VOC dataset, we wanted to show what improvements others were able to make on similar tasks. It also shows why the improvement in mAP for the QUAT models was more significant than the raw numbers.

The YOLOv3-tiny QUAT (1.0) and (2.0) models mostly performed better than their regular counterpart, but the adversarial training was not as successful as it was for the YOLOv3 QUAT(1.0) or the YOLOv3 and Faster RCNN QUAT(2.0) models. We hypothesize that this is because the YOLOv3-tiny model is smaller and has less weights to train on and so the extra data was less beneficial. Instead of getting a 17% increase, we got an improvement between $1-2\%$. While this may seem very low, it is similar to the improvements that were obtained in the aforementioned paper. Also, the YOLOv3-tiny models were only able to achieve $20-25\%$ mAP on clean data. If we were to make that our baseline for the YOLOv3-tiny models, the improvement is much more significant, than if our baseline were set at $75-100\%$. This also makes the YOLOv3 and Faster RCNN models improvement more significant because even though the baseline is at around 45% and $60-70\%$ mAP, it improved by up to 17% and up to 20%.

V. Discussion and Conclusions

In this paper, we presented an evaluation framework for robust object detection as well as a defense technique for the edge and cloud level. We went through two different iterations of this process for our defense with our second iteration being much more resource and time-friendly. This is an important area to explore given the rising number of edge-based applications that could benefit from robust machine learning solutions.

The two techniques we described both had promising results. The YOLO models were in general faster at processing than the QUAT YOLO models. However, we saw the opposite occur for YOLOv3-tiny models, and there was not much difference for the Faster R-CNN models. In most cases, the QUAT approach improved the adversarial accuracy. The very promising result was that QUAT (2.0) actually increased clean accuracy as well as improved adversarial accuracy. This is not usually the case with adversarial training. We hypothesize that this is due to the fact that instead of augmenting the training set with more data, we swapped out clean training images for a diverse set of adversarial images. One potential issue with the augmentation approach could be the model over-fitting to the training set and this is why clean accuracy dropped.

Our work serves as a step towards deploying adversarially robust deep learning object detection models at the edge as well as the cloud.

A. Future Work

For future work, we plan to evaluate our QUAT approach on other Object Detection models and a range of edge devices. Further, we plan to define an approach to robustly and efficiently offload computation when necessary. The offload could serve as a robustness check because of the fact that the smaller edge-based models lack the performance of the cloud-based models on clean data as well as adversarial data. We also plan to implement other object detection defenses and see how they compare to QUAT against a variety of attacks either physically or through software. The ultimate goal of further research will be to construct a closed loop system for edge-based deep learning that assures robustness and latency demands are met. We plan to start testing on a larger scale with real-life situations like a connected neighborhood, autonomous driving scenario, navigational aid to the visually impaired, or AI-assisted augmented reality for smart manufacturing.

B. Code

Code will be made available on github at https://github.com/canadyre/quat.

Acknowledgments

This work was supported in part by the US NSF's Smart and Connected Communities Program under Award 1952029. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsor.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[2] R. Huang, J. Pedoeem, and C. Chen, "Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 2503–2510.

[3] X. Zhou, R. Canady, S. Bao, and A. Gokhale, "Cost-effective hardware accelerator recommendation for edge computing," in *3rd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 20)*, 2020.

[4] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2020.

[5] J. Yang, A. Boloor, A. Chakrabarti, X. Zhang, and Y. Vorobeychik, "Finding physical adversarial examples for autonomous driving with fast and differentiable image compositing," *arXiv preprint arXiv:2010.08844*, 2020.

[6] A. D. Lascorz, S. Sharify, I. Edo, D. M. Stuart, O. M. Awad, P. Judd, M. Mahmoud, M. Nikolic, K. Siu, Z. Poulos, and A. Moshovos, "Shapeshifter: Enabling fine-grain data width adaptation in deep learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 28–41.

[7] X. Ji, Y. Cheng, Y. Zhang, K. Wang, C. Yan, W. Xu, and K. Fu, "Poltergeist: Acoustic adversarial machine learning against cameras and computer vision," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 160–175.

[8] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. L. Yuille, "Adversarial examples for semantic segmentation and object detection," *Arxiv preprint arxiv:1703.08603*, vol. abs/1703.08603, 2017.

[9] Y. Li, X. Bian, and S. Lyu, "Attacking object detectors via imperceptible patches on background," *Arxiv preprint arxiv:1809.05966*, 2018.

[10] K.-H. Chow, L. Liu, M. Loper, J. Bae, M. Emre Gursoy, S. Truex, W. Wei, and Y. Wu, "Adversarial objectness gradient attacks in real-time object detection systems," in *IEEE International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications*, 2020, pp. 263–272.

[11] K.-H. Chow, L. Liu, M. E. Gursoy, S. Truex, W. Wei, and Y. Wu, "Understanding object detection through an adversarial lens," in *European Symposium on Research in Computer Security*. Springer, 2020, pp. 460–481.

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[13] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *Arxiv preprint arxiv:1506.01497*, 2015.

[14] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *Arxiv preprint arxiv:1804.02767*, 2018.

[15] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[16] OpenDataCam, "opendatacam/opendatacam." [Online]. Available: https://github.com/opendatacam/opendatacam

[17] Z. Xu, H. S. Shah, and U. Ramachandran, "Coral-pie: A geo-distributed edge-compute solution for space-time vehicle tracking," in *Middleware '20: Proceedings of the 21st International Middleware Conference*, 2020, pp. 400–414.

[18] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Arxiv preprint arxiv:1712.03141*, 2017.

[19] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," *Arxiv preprint arxiv:1708.06131*, 2017.

[20] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[21] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," *Arxiv preprint arxiv: 1511.04599*, 2015.

[22] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," *Arxiv preprint arxiv:1608.04644*, 2016.

[23] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," 2017.

[24] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," vol. Arxiv preprint arxiv:1605.07277, 2016.

[25] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2019.

[26] R. G. Lopes, D. Yin, B. Poole, J. Gilmer, and E. D. Cubuk, "Improving robustness without sacrificing accuracy with patch gaussian augmentation," *Arxiv preprint arxiv:1906.02611*, 2019.

[27] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. L. Yuille, "Mitigating adversarial effects through randomization," *Arxiv preprint arxiv:1711.01991*, 2017.

[28] E. Raff, J. Sylvester, S. Forsyth, and M. McLean, "Barrage of random transforms for adversarially robust defense," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[29] C. Xie, Y. Wu, L. v. d. Maaten, A. L. Yuille, and K. He, "Feature denoising for improving adversarial robustness," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 501–509.

[30] Z. Hu and Z. Zhong, "Towards practical robustness improvement for object detection in safety-critical scenarios," in *Deployable Machine Learning for Security Defense*, G. Wang, A. Ciptadi, and A. Ahmadzadeh, Eds., 2020.

[31] H. Zhang and J. Wang, "Towards adversarially robust object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[32] X. Chen, C. Xie, M. Tan, L. Zhang, C.-J. Hsieh, and B. Gong, "Robust and accurate object detection via adversarial learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2021.

[33] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. J. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," *CoRR*, vol. Arxiv preprint arxiv:1902.06705, 2019.

[34] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, "Augmix: A simple data processing method to improve robustness and uncertainty," 2020.