Veiled from Above: Adversarial Attacks on Satellite Imagery Detectors

Andrew Milich
Stanford University
amilich@stanford.edu

June 5, 2019

Abstract

We studied the sensitivity of satellite imagery object detection models to adversarial input images. We began by training the object detection model Tiny-YOLOv3 on the dataset Cars Overhead With Context (COWC), which contains over 27,000 distinct full-color satellite images of cars with bounding boxes. After using transfer-learning to train our object detector, we studied its susceptibility to three types of adversarial inputs: Random noise, images produced using the fast gradient sign method (FGSM), and an adversarial patch. These attacks were inspired by seminal and state-of-the-art research in adversarial training. Unsurprisingly, we found that random noise - our control experiment - did not significantly reduce model accuracy, as measured by mean average precision. In contrast, FGSM attacks and our adversarial patch did undermine model performance without significantly changing the content of an input image. However, given results in the literature, we expect that these attacks can be further refined to be far more effective. Ultimately, as this area of object detection has not been deeply studied by existing literature, our results present clear applications to attacking and defending satellite imagery detectors.

1. Introduction

Recent research has demonstrated how deep learning models for object detection may be highly sensitive to small perturbations in input images.[1] For example, some papers have reduced object detection performance by over 50% by only modifying a single pixel in an input image.[2] Recent research papers have proposed a variety of attack mechanisms for undermining deep learning models, from changing a single pixel in an input image to recoloring an image in the direction of the gradient of the loss function (an attack known as FGSM, or the fast gradient sign method).[3] We sought to study this problem in the context of satellite imagery object detection. Are deep learning models for detecting objects in satellite images sensitive to these attacks?

We were particularly interested in this project due to its application of adversarial techniques to a problem not significantly studied in the literature; while past projects and papers have examined adversarial attacks on facial recognition and object detection in datasets such as ImageNet or PASCAL VOC,[4] far fewer research groups have examined this problem in satellite imagery.[5] Furthermore, satellite imagery object detection represents an important application of deep learning to national security and humanitarian work. Data on shipping and travel patterns has been widely used by governments and non-governmental organizations to track economic activity, population movement, and conflict.[6] As a result, it is important for humanitarian and governmental organizations to understand these models' weaknesses.

Prior to this project, I did not have any experience using PyTorch or training object detection models; by applying cutting-edge adversarial techniques to a relatively unstudied area of object detection, I hoped to make a novel and timely contribution to understanding the capabilities and vulnerabilities of deep learning models while also gaining the requisite skills for performing deep learning research.

1.1. Literature Review

The field of generating adversarial examples is relatively nascent. Ian Goodfellow's 2014 paper "Explaining and Harnessing Adversarial Examples" outlined the fast gradient sign method (FGSM) for perturbing sample images using the data gradient of a model's loss function; this paper also relied on random perturbations as control experiments for comparing performance. Since then, researchers have released open-source software for generating adversarial images, such as the library Cleverhans.[7] In our project, we began by implementing baseline and adversarial techniques introduced in Goodfellow's 2014 paper, such as adding random noise and performing FGSM attacks.

1. Goodfellow, Shlens, and Szegedy 2014.
2. Su, Vargas, and Sakurai 2019.
3. Su, Vargas, and Sakurai 2019; Goodfellow, Shlens, and Szegedy 2014; Madry et al. 2017.
4. Deng et al. 2009; Everingham et al. 2010.
5. Milich and Karr, n.d.
6. Doshi, Basu, and Pang 2018.
7. "tensorflow/cleverhans: An adversarial example library for constructing attacks, building defenses, and benchmarking both" 2019.

The paper "Adversarial Patch," which was published by researchers at the MIT Media Lab in 2017, demonstrated how a physical patch could be created to undermine object detection models. The authors used a technique called an "Expectation over Transformation" (EoT) attack that requires maximizing an objective function to achieve misclassifications across different orientations of a physical object. When placed alongside a 3D-printed turtle, the patch caused the VGG-16 object detection model to miscategorize the turtle as a rifle from multiple angles.[8] We discuss the optimization problem in an EoT attack in Equation (2). A more recent paper titled "Fooling automated surveillance cameras: adversarial patches to attack person detection" used similar techniques to confuse person detection models.[9] Other published papers have examined techniques for undermining state-of-the-art real-time object detectors, such as R-CNN and YOLO, using expectation over transformation and other attack mechanisms.[10]

In preparation for this project, we also examined two recent deep learning papers on state-of-the-art real-time object detection: "YOLOv3: An Incremental Improvement" and "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks."[11] YOLOv3 divides images into cells that are used to generate bounding boxes for potential objects; the most recent version of the network also relies on stacked 3×3 filter convolutions for feature extraction. R-CNNs rely on a convolutional network to propose regions that are subsequently used as inputs to another object detection model. Ultimately, we chose to test our attack mechanisms on Tiny-YOLOv3 due to the model's recency, relatively fast training speeds, efficient test-time performance, and the availability of an easily extendable implementation in PyTorch.[12]

2. Methods

This section discusses our overall methodology for training our object detector and generating adversarial examples.

2.1. Object detection

The YOLO - "You Only Look Once" - family of object detectors provide high accuracy and notably fast predictions. YOLO's uniquely swift performance stems from its ability to make predictions for multiple objects using only one pass through the entire network. The original YOLOv1 network divided images into a grid and predicted bounding boxes in each distinct cells;[13] the YOLOv1 network's overall architecture resembled Google LeNet and used 24 convolutional layers.[14] YOLOv2 and YOLOv3 provided better performance and new algorithms for detecting object bounding boxes; in YOLOv3, the network was significantly expanded to include a feature extractor with 53 convolutional layers.[15] YOLOv3 is a fully convolutional network that does not use pooling layers. Instead, it relies on convolutions with a stride of two for downsampling and 1x1 convolutions to make predictions. YOLOv3 also uses "that newfangled residual network stuff" (i.e. residual connections between groups of convolutional layers) to prevent loss of low level features.[16]

Before discussing our performance metrics for training YOLO on satellite imagery, we provide a brief overview of the YOLOv3 loss function. Because of length constraints, we do not provide the full mathematical formula for the loss function; this is available in the original YOLOv1 paper.[17] The YOLO loss function sums three terms:

$$\mathbb{L} = L_{classification} + L_{localization} + L_{confidence}$$

Classification loss measures the squared error of the probability of each class when an object is detected in a given cell of an image. **Localization loss** computes the squared error of predicted bounding box positions and shapes. **Confidence loss** sums the squared error of the model's confidence that a given object is or is not in a given cell of an image.[18]

To train our satellite imagery object detector, we used an open-source PyTorch implementation of YOLOv3[19] and a set of pretrained weights for Tiny-YOLOv3 on the object detection dataset COCO. Because of our dataset's relatively small size and constriants on time and computing hardware, we chose to use the Tiny YOLOv3 model, which is significantly smaller than the full YOLOv3 network. This enabled us to more quickly prototype and test attack mechanisms.

2.1.1 Performance metrics

Intersection over union (IoU) and **average precision** are standard metrics for measuring object detection performance. Given a model's predicted object bounding box and a known ground-truth bounding box, we can score the predicted bounding box using its IoU with the ground-truth box:

8. Brown et al. 2017.
9. Thys, Van Ranst, and Goedeme 2019.
10. Lu, Sibai, and Fabry 2017; Chen et al. 2018.
11. Redmon and Farhadi 2018; Ren et al. 2015.
12. "michhar/pytorch-yolo-v3-custom: A PyTorch implementation of the YOLO v3 object detection algorithm for training on custom data with video demo." 2019.
13. Redmon et al. 2016.
14. Szegedy et al. 2015.
15. Redmon and Farhadi 2018.
16. Redmon and Farhadi 2018; "How to Implement a YOLO (v3) Object Detector from Scratch in PyTorch: Part 1" 2019.
17. Redmon et al. 2016.
18. "Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3" 2019.
19. "michhar/pytorch-yolo-v3-custom: A PyTorch implementation of the YOLO v3 object detection algorithm for training on custom data with video demo." 2019.

$$\text{IoU}(bbox_1, bbox_2) = \frac{\text{intersection area}(bbox_1, bbox_2)}{\text{union area}(bbox_1, bbox_2)}$$

Note that if the two bounding boxes are exactly the same, their IoU is exactly 1; if they are completely disjoint, their IoU is 0. When evaluating object detection models, an IoU threshold is chosen for differentiating true from false positive. The value of 0.5 IoU is common in object detection literature.[20] However, we chose to test model performance on IoU values of 0.3 and 0.4 because a significant quantity of our train and test images had irregularly positioned or overlapping boxes.[21] Furthermore, unlike many object detection datasets, where objects may occupy a large portion of an image, a single car may only occupy one percent of a training image. Given the above formulation for IoU, we can determine if a prediction is correct if it has IoU with a ground truth bounding box that is greater than a certain threshold.

Object detection networks are frequently evaluated by their **average precision**. First, predictions from a given dataset are ordered by decreasing confidence. Then, for each prediction, running values of precision and recall are calculated; precision equals the number of true positives divided by true and false positives, and recall equals the number of true positives divided by the number of true positives plus false negatives. Given precision and recall values for the model's output on the testing dataset, average precision equals the area under the precision recall curve.[22] The **mean average precision** (mAP) is the average precision avereged across all classes; in our application of detecting one class - sedan cars - mAP is equal to AP and is used interchangeably.

2.2. Generating adversarial examples

In this section, we discuss how our three different categories of adversarial input images were generated. Within the field of generating adversarial examples, attack mechanisms are divided into two categories: Whitebox and blackbox attacks. While whitebox attacks assume an attacker has access to the internal parameters of a given model, an adversary may only observe a model's output when performing blackbox attacks.[23]

1. **Random noise**: Our baseline or control attack mechanism involved sampling random noise from a Gaussian distribution and adding it to our input image. Random noise has been previously used as a control experiment in existing literature on generating adversarial examples.[24] Random noise is a blackbox attack that does not require access to the model.

2. **FGSM attacks**: After running an image x through the model, an FGSM attack perturbs the image x by a certain amount in the direction of the sign of the data gradient, i.e.:[25]

$$x_{adv} = x + \epsilon \, sign(\nabla_x J(\theta, x, y)) \tag{1}$$

   FGSM is a whitebox attack mechanism intended to cause misclassifications; it requires access to the gradient of the model's loss function $J(\theta, x, y)$.

3. **Adversarial patch**: The 2017 paper "Synthesizing Robust Advesarial examples" paper proposes a method known as an "Expectation over Transformation" (EOT) attack that is capable of generating adversarial examples that undermine an object detector across different orientations and viewpoints.[26] The EOT attack represents the following optimization problem, where x is the original input image, x' is the perturbed adversarial image, $t(x)$ is a transformation of the input x, y_t is the desired target class, and $P(y|t(x))$ is the probability that transformed image $t(x)$ is classified as y. This optimization attempts to maximize the probability that the transformed image x' is classified as a target class y_t subject to the constraint that x' can only be perturbed a limited amount.

$$\begin{aligned} &\arg\max_{x'} \quad \mathbb{E}_{t\sim T}\left[\log P\left(y_t | t\left(x'\right)\right)\right] \\ &\text{subject to } \mathbb{E}_{t\sim T}\left[d\left(t\left(x'\right), t(x)\right)\right] < \epsilon \end{aligned} \tag{2}$$

   This method is the most complex attack mechanism tested in this project. Figure 1 provides an overview of the training process for creating the adversarial patch.

   The patch is initialized randomly. Then, using our validation dataset, we rotate and translate the patch before applying it to an image. We then compute the YOLO loss for the given patch and perturb the patch in the direction of the model's loss gradient.

20. Hui 2019.
21. It also required a nontrivial amount of time to evaluate mAP on distinct datasets, so we chose to use metrics that provided more insight into our attack effectiveness.
22. Hui 2019.
23. Researchers have demonstrated how an adversary may use a model's output to train a comparable model in the blackbox scenario; this enables the attacker to approximate the unknown model's gradient and loss. (Papernot et al. 2017)
24. Goodfellow, Shlens, and Szegedy 2014.
25. Goodfellow, Shlens, and Szegedy 2014.
26. Athalye et al. 2017.

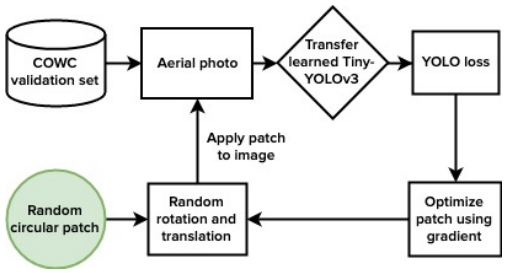

Figure 1: Training procedure for generating an adversarial patch.

We trained our patch using a significantly modified open-source implementation of the training procedure used in the 2017 paper "Adversarial Patch."[27] Our patch is set to cover 5% of the input image, which results in a patch that is roughly 100×100 pixels. Figure 8 displays the results of training our patch on the validation dataset.

3. Dataset and features

Multiple publicly accessible datasets exist for training object detection models on satellite imagery. Popular datasets include "Cars Overhead With Context" (COWC), "xView: Objects in Context in Overhead Imagery," and "DOTA: Large-scale Dataset for Object Detection in Aerial Images."[28] We chose to train our model on the COWC dataset, which was released by Lawrence Livermore National Laboratory. The COWC dataset includes over 27,000 satellite images with a total of 61,000 bounding box annotations. We selected this dataset because it provided a sufficiently large number of training images with relatively few classes; while DOTA and xView contain annotations of numerous different objects, including planes, helicopters, swimming pools, bridges, and basketball courts, the COWC dataset is designed specifically for detecting cars.

We used a version of the COWC dataset called "COWC-M" which formats ground-truth labels in a manner that matches the expected YOLO annotation format of:[29]

```
<object-class> <x> <y> <width> <height>
```

Images in the COWC-M dataset are 256×256 pixels. In Figure 2, we provide an example of a single image in the COWC-M dataset with one ground-truth bounding box around a single car:

Figure 2: Example aerial photo from the COWC dataset. One blue bounding box surrounds the single car in the image.

For clarity, we provide a single ground-truth label below. Note that x and y) (i.e. the second and third numbers in the label) represent the center of the bounding box.

```
1 0.07421875 0.4140625 0.125 0.125
```

The above label specifies a bounding box with width and height equal to 0.125×256 centered at $(x, y) = (0.07421875 \times 256, 0.4140625 \times 256)$.

It is important to note that we observed a general lack of consistency in the size of bounding boxes and the position of cars within them. Although most boxes are squares, the cars are frequently not positioned exactly at the center of boxes, and, in many examples, the hood or rear of the car is cut off. Because of these irregularities in ground-truth bounding boxes, we chose to measure mAP using thresholds of 0.3 and 0.4. These metrics also seemed appropriate given how small cars are in each training photo; while objects in the COCO dataset may occupy one half or one third of an input image, a car may only occupy a small percentage (as small as one percent) of a COWC photo.

3.1. Preprocessing

We performed a limited number of preprocessing steps to format the COWC dataset for training with YOLOv3. YOLO does not require image normalization or transformation; as a result, we only focused on aggregating and formatting the COWC images for training with YOLO. Because the YOLO network utilizes only convolutional layers, it can accept inputs of different dimensions. In training the basic object detector, we performed a series of data augmentation steps, including random recolorings and horizontal flips. When training our adversarial patch, we implemented our own library of data augmentation functions for rotating and flipping images.

To construct our training, validation, and test datasets, we aggregated the distinct COWC-M imagery datasets from Toronto (Canada), Utah (USA), and Selwyn (New Zealand).[30] We used a 0.8-0.1-0.1 train-validation-test split

27. "jhayes14/adversarial-patch: PyTorch implementation of adversarial patch" 2019.
28. Mundhenk et al. 2016; Lam et al. 2018; Xia et al. 2018.
29. "Specific format of annotation Issue 60 AlexeyAB/Yolo_mark" 2019.
30. Mundhenk et al. 2016.

Figure 3: Model output superimposed with ground truth bounding boxes. Ground truth boxes are in green; model output is in red.

across a randomly shuffled aggregate dataset from all locations combined. This ensured our model could both train and test on cars in different contexts from rural and urban environments. Given this configuration, we had almost 19000 training samples, roughly 2400 validation samples, and approximately 2400 held-out test samples.

4. Results

4.1. Object detection results

We optimized our hyperparameters using our validation set. Although one published YOLO model was trained using SGD and weight decay, we observed better training performance (i.e. faster convergence) training Tiny-YOLOv3 using the Adam optimizer.[31] We also used a decaying learning rate starting at 0.001, and a batch size of 4 images.[32][33] Because we chose to perform transfer-learning on Tiny-YOLO using pretrained weights from the COCO dataset, we ultimately saw good performance unfreezing the last three layers of the network. Figure 3 provides one sample image of our model's output superimposed with ground truth bounding boxes from the COWC dataset.

Our object detection results are reported in Table 1. Because of the wide variance in the size and positioning of car bounding boxes in the COWC-M dataset (this is discussed further in Section 3), we chose to primarily measure performance using AP_{30} and AP_{40} instead of AP_{50} and AP_{75}, which are metrics used in the YOLOv3 paper. We also selected these metrics because the performance of Tiny-YOLOv3 is generally reported to be considerably lower than that of YOLOv3.[34] In many training images, bounding boxes appear to arbitrarily extend well past the cars inside them, which may explain why our model has relatively low AP_{50} scores.

31. Redmon and Farhadi 2017.
32. Note that the full YOLOv3 network is generally trained with a much larger batch size.
33. Kingma and Ba 2014.
34. "YOLO: Real-Time Object Detection" 2019.

Our model's relatively good performance on the Selwyn images in the training set suggests that the model is not overfitting. Instead, its lower performance on the Utah and Toronto data suggests that we could have further optimized our training procedure or used the full YOLOv3 model.

Dataset	mAP_{30}	mAP_{40}
All test	0.21	0.16
Utah (1247 images)	0.17	0.14
Selwyn (232 images)	0.48	0.34
Toronto (882 images)	0.23	0.20

Table 1: Per-dataset object detection results.

4.2. Random noise results

Table 2 provides the results of adding norm-constrained Gaussian random noise to input images.

Dataset	mAP_{30}	mAP_{40}
All test	0.20	0.14
Utah (1247 images)	0.14	0.11
Selwyn (232 images)	0.46	0.23
Toronto (882 images)	0.21	0.15

Table 2: Model output on images perturbed with random noise.

4.3. FGSM results

Table 3 provides the results of performing an FGSM attack on input images. Note that we constrained the norm of the image perturbation to be approximately equal to that of the random noise perturbation performed above.

Dataset	mAP_{30}	mAP_{40}
All test	0.14	0.07
Utah (1247 images)	0.09	0.03
Selwyn (232 images)	0.19	0.13
Toronto (882 images)	0.22	0.09

Table 3: Model output on images perturbed with an FGSM attack.

4.4. Adversarial patch results

Table 4 reports the results of running our object detector on images with our adversarial patch applied. The patch is applied to a random location in the image at a random rotation (as is done during the EoT optimization procedure).[35]

35. We also considered using a saliency map from YOLOv3 trained on COCO to choose a location for the patch, but existing literature uses a random location.

Dataset	mAP_{30}	mAP_{40}
All test	0.13	0.09
Utah (1247 images)	0.11	0.06
Selwyn (232 images)	0.23	0.12
Toronto (882 images)	0.16	0.08

Table 4: Model output on images perturbed with an adversarial patch that occupies 5% of the image.

5. Discussion

Although our model's AP_{40} results on unperturbed training images are significantly lower than published results of YOLOv3 on the COCO dataset (mAP_{50} of 57.9[36]), multiple factors explain this disparity. One principal difference is that we chose to use Tiny-YOLOv3 instead of the full YOLOv3 model. Even tough detailed results on Tiny-YOLOv3 could not be found in published research, one website lists the mAP for Tiny-YOLOv3 as 33.1, which is more similar to our results.[37] This is unsurprising as Tiny-YOLOv3 is significantly less complex than YOLOv3. Another explanatory factor is the relatively small size of our training datset; while the COCO dataset used to train and test YOLOv3 has over 330,000 images, our satellite imagery dataset, which was used for transfer learning, has only 27,000 images.[38] Generally, these differences appear to be largely due to our use of the much smaller Tiny-YOLOv3 network.

It is both notable and relatively unsurprising that our model achieves very strong results on the raw Selwyn dataset (mAP_{30} of 0.48 and mAP_{40} of 0.34) and weaker performance on the other imagery datasets. The Selwyn dataset consists of a relatively homogeneous group of photos of cars on green, rural backgrounds. Over 58% of photographs in the Selwyn dataset have only a single ground-truth example; another 20% have only two ground-truth bounding boxes. As a result, it was likely far easier for our model to identify and construct bounding boxes for a single car with a green or dirt background. Our model does not perform as well on the Toronto and Utah datasets; in both datasets, cars frequently appear against grey or road backgrounds, and numerous cars may appear in a single input image. Given our limited training dataset and use of the Tiny-YOLOv3 network, our model may simply not have learned to identify cars as well in these contexts.

Random noise achieved limited results as an attack mechanism. In almost all instances, model performance decreased - unsurprisingly - but was not significantly reduced. On the Selwyn dataset, mAP_{30} remained roughly

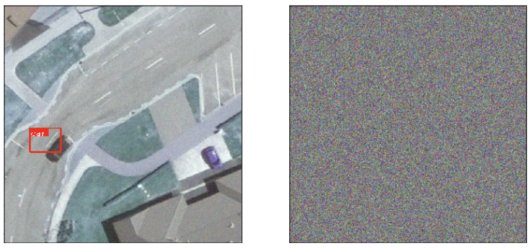

Figure 4: Example of our model's output on an image perturbed by random noise. On the left is the perturbed image, and the right is the random noise added. For better viewing, the Gaussian noise on the right is scaled to occupy the color range from 0 to 1.

Figure 5: Model output on the same image as in Figure 4 but with *no random perturbation added*. The bounding box is more horizontally centered around the car.

the same while mAP_{40} decreased, perhaps suggesting that random noise impaired the quality of our model's bounding boxes but did not significantly impede object identification. The observation that AP_{40} decreased for all datasets also supports this hypothesis. The example photos provided in Figures 4 and 5 provide additional qualitative evidence to support this theory about our model's performance under random noise perturbations. The bounding box on the left of Figure 4 is offset from the center of the car (note that the purple car in the image is undetected). In the same image *without random perturbations*, which is depicted in Figure 5, the bounding box is more properly centered around the car. As a result, the bounding box in Figure 4 likely has a lower IoU score than the bounding box in Figure 5.

FGSM attacks provided a more effective mechanism for undermining our object detector. On the Selwyn dataset, mAP_{40} decreased to 0.19, which is lower than when images were perturbed with random noise. Figures 6 and 7 showcase examples of our model's output on the same training image with and without FGSM perturbations. Although the perturbed image displayed on the left of Figure 6 is imperceptibly different from the image in Figure 7, the model fails to identify the single car pictured. It is notable

36. Redmon and Farhadi 2018.
37. "YOLO: Real-Time Object Detection" 2019.
38. Lin et al. 2014.

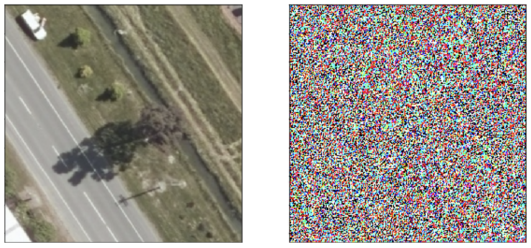

Figure 6: Example of our model's output on an image perturbed by an FGSM attack. On the left is the modified image, and the right is the FGSM perturbation scaled to occupy the color range from 0 to 1. The model fails to detect the white car in the top left of the photo, which it does detect in the raw image as displayed in Figure 7.

Figure 7: Model output on the same image as in Figure 6 but with no FGSM perturbation. The model clearly identifies the car correctly.

that FGSM attacks were particularly effective on the Utah dataset; this may be because the Utah dataset represented over half of the test images, so perhaps FGSM attacks - which use the model's gradient - somewhat overfit to these images.

Our adversarial patch represented a surprisingly effective attack algorithm. Unlike random noise and FGSM attacks, the patch is not imperceptible; although it only covers five percent of the input image, the patch is easily visible. Furthermore, in contrast to an FGSM attack, the adversarial patch does not require access to the model's test-time gradient. On the Selwyn dataset, the patch resulted in an mAP_{30} of 0.23 and an mAP_{40} of 0.12; these numbers are by far the best performance on images perturbed with a patch. This result - that our patch was not particularly effective on Selwyn images - is unsurprising for multiple reasons: Our model performed the best on the unperturbed Selwyn dataset, and the Selwyn dataset is also significantly smaller (232 images) than the Utah (1247 images) and Toronto (882 images) ones. As a result, our validation dataset - which was used to train the patch - contained far fewer of these examples. Thus, the patch was not adversarially trained on as many Selwyn images.

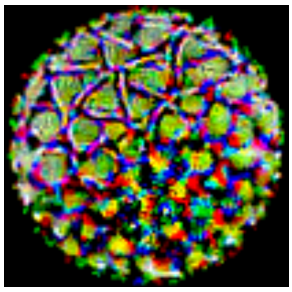

Figure 8: Example of our adversarial patch superimposed against a black background. As in Equation (2), the patch was trained using transformations that included random translations in an input image and random rotations.

Figure 9: On the left, our model detects the edge of the red car in the driveway. On the right, with the patch added, the model outputs no confident prediction.

We trained multiple adversarial patches on our validation set of images. As in the open-source implementation we used,[39] we initialized the patch to be a random single color. Then, using the validation set of images, we optimized the patch across different rotations and translations in input images. Figure 8 displays one of the patches trained from the validation set. Although it is difficult to speculate about the features that are present on the patch, it is possible that the lines and green regions reflect features learned during the training process to signify the presence or absence of a car. Because patches are randomly initialized, rotated, and translated, other patches trained on the validation set appear very different.

Ultimately, although our patch and FGSM attacks were effective in multiple cases, we initially expected them to be far more capable of undermining our object recognition model. In "Explaining and Harnessing Adversarial Exam-

39. "jhayes14/adversarial-patch: PyTorch implementation of adversarial patch" 2019.

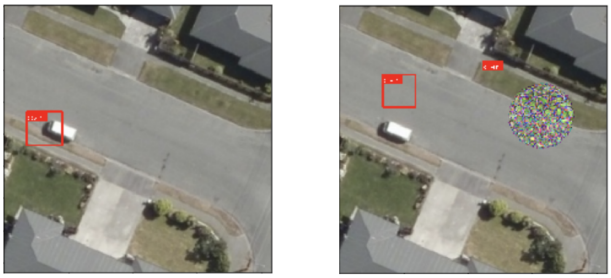

Figure 10: On the left, our model accurately identifies the car. On the right, with the patch added, our model outputs multiple false positives.

ples," an FGSM attack yields over 85% misclassification rate on the MNIST with $\epsilon = 0.1$. Even though we made smaller perturbations to input images by setting $\epsilon = 0.05$, and detecting cars in the COWC dataset is significantly more complex than MNIST classification, we initially expected FGSM attacks to yield significantly lower mAP values for our network. The latest adversarial patch literature has also suggests that our attacks could have been far more effective; one published paper reports that a single patch could cause YOLO mAP to drop to below 1%.[40] More experiments - such as using only Utah or Toronto images for training and testing - could lend insight into these disparities.

6. Conclusion and future work

Our results provide compelling insights into how adversarial attacks could be used to undermine object detectors trained on satellite imagery. Random noise was not particularly effective in reducing our classifier's mAP, thus suggesting that our object detector is robust to minor, non-localized perturbations in input images. However, in some cases, random noise caused the classifier to predict bounding boxes that had a lower IoU with ground truth labels. In contrast, FGSM attacks unsurprisingly reduced our model's mAP below the values reported when just using random noise. However, FGSM is a whitebox attack that requires knowing or approximating the model's gradient. Although both FGSM and random noise attacks yield relatively imperceptible modifications to input images, FGSM attacks proved more effective.

Applying an adversarial patch represented our most sophisticated attack mechanism and required training across a validation set of over two thousand images. Random variation in initialization and training yielded patches with vastly different appearances. Even without access to the model's test-time gradient, this attack mechanism did successfully reduce our models' mAP to a greater extent than our baseline attack using random noise. Yet, using an adversarial patch as a real-world blackbox attack mechanism may be impractical; creating the patch requires access to the model's gradient, and the patch itself is easily spotted during visual inspection. However, as academic literature continues to study object detectors' susceptibility to adversarial attacks, our work demonstrates how some models could be compromised using this technique.

Given these experiments in undermining our satellite imagery object detector, we identified multiple promising areas of future work in this project. Instead of simply training a circular patch applied to an image, we were interested in adversarially training a camouflage patch that could be intelligently applied to objects, such as the surface of cars. This application presents clear national security significance as adversarial camouflage could potentially be used to impede detection of cars or other objects. Furthermore, instead of testing and optimizing our attacks on Tiny-YOLOv3, we are interested in experimenting with the full YOLOv3 network (or potentially R-CNN models as well). Finally, applying algorithms for defending YOLOv3 from adversarial attacks or detecting adversarial perturbations represents another compelling area for future research. Thus, there are numerous exciting avenues for additional projects stemming from this work.

7. Contributions and Acknowledgements

Andrew Milich completed the entirety of this project. During the proposal and implementation process, CS 231n TAs Andrew Han, William Shen, and Saahil Agrawal provided helpful direction and guidance. Initially, I focused on generating adversarial examples in the context of detecting ship positions on the ocean (this was the focus of my project update). However, I subsequently decided to transition to using a more complex dataset (COWC) and object detection model (YOLOv3).

All open-source and customized deep learning models were implemented using PyTorch.[41] For data processing, creating an adversarial patch, and image manipulation, we also used Scipy and Numpy.[42] We used an open-source PyTorch implementation of YOLOv3[43] for training an object detector on the COWC dataset.[44] When creating our adversarial patch, we used elements of an open-source PyTorch implementation;[45] however, we also wrote a customized method for transforming the patch and applying it to images.

40. Liu et al. 2019.

41. "PyTorch" 2019.

42. Jones, Oliphant, and Peterson 2014; Developers 2013.

43. "michhar/pytorch-yolo-v3-custom: A PyTorch implementation of the YOLO v3 object detection algorithm for training on custom data with video demo." 2019.

44. Mundhenk et al. 2016.

45. "jhayes14/adversarial-patch: PyTorch implementation of adversarial patch" 2019.

References

Athalye, Anish, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. 2017. "Synthesizing robust adversarial examples." *arXiv preprint arXiv:1707.07397.*

Brown, Tom B, Dandelion Mane, Aurko Roy, Martin Abadi, and Justin Gilmer. 2017. "Adversarial patch." *arXiv preprint arXiv:1712.09665.*

Chen, Shang-Tse, Cory Cornelius, Jason Martin, and Duen Horng Polo Chau. 2018. "ShapeShifter: Robust Physical Adversarial Attack on Faster R-CNN Object Detector." In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases,* 52–68. Springer.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. "Imagenet: A large-scale hierarchical image database." In *2009 IEEE conference on computer vision and pattern recognition,* 248–255. Ieee.

Developers, NumPy. 2013. "NumPy." *NumPy Numpy. Scipy Developers.*

Doshi, Jigar, Saikat Basu, and Guan Pang. 2018. "From Satellite Imagery to Disaster Insights." *arXiv preprint arXiv:1812.07033.*

Everingham, Mark, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2010. "The pascal visual object classes (voc) challenge." *International journal of computer vision* 88 (2): 303–338.

Goodfellow, Ian J, Jonathon Shlens, and Christian Szegedy. 2014. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572.*

"How to Implement a YOLO (v3) Object Detector from Scratch in PyTorch: Part 1." 2019. Accessed June 1. `https://www.kdnuggets.com/2018/05/implement-yolo-v3-object-detector-pytorch-part-1.html`.

Hui, Jonathan. 2019. "mAP (mean Average Precision) for Object Detection." *Medium.* Accessed June 1. `https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173`.

"jhayes14/adversarial-patch: PyTorch implementation of adversarial patch." 2019. Accessed June 1. `https://github.com/jhayes14/adversarial-patch`.

Jones, Eric, Travis Oliphant, and Pearu Peterson. 2014. "{SciPy}: Open source scientific tools for {Python}."

Kingma, Diederik P, and Jimmy Ba. 2014. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980.*

Lam, Darius, Richard Kuzma, Kevin McGee, Samuel Dooley, Michael Laielli, Matthew Klaric, Yaroslav Bulatov, and Brendan McCord. 2018. "xview: Objects in context in overhead imagery." *arXiv preprint arXiv:1802.07856.*

Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. "Microsoft coco: Common objects in context." In *European conference on computer vision,* 740–755. Springer.

Liu, Xin, Huanrui Yang, Ziwei Liu, Linghao Song, Hai Li, and Yiran Chen. 2019. "DPATCH: An Adversarial Patch Attack on Object Detectors." *AAAI Workshop on Artificial Intelligence Safety (SafeAI).* Accessed June 1, 2019. `https://arxiv.org/abs/1806.02299`.

Lu, Jiajun, Hussein Sibai, and Evan Fabry. 2017. "Adversarial examples that fool detectors." *arXiv preprint arXiv:1712.02494.*

Madry, Aleksander, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. "Towards deep learning models resistant to adversarial attacks." *arXiv preprint arXiv:1706.06083.*

"michhar/pytorch-yolo-v3-custom: A PyTorch implementation of the YOLO v3 object detection algorithm for training on custom data with video demo." 2019. Accessed June 1. `https://github.com/michhar/pytorch-yolo-v3-custom`.

Milich, Andrew, and Michael Karr. n.d. "Eluding Mass Surveillance: Adversarial Attacks on Facial Recognition Models."

Mundhenk, T Nathan, Goran Konjevod, Wesam A Sakla, and Kofi Boakye. 2016. "A large contextual dataset for classification, detection and counting of cars with deep learning." In *European Conference on Computer Vision,* 785–800. Springer.

Papernot, Nicolas, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. "Practical black-box attacks against machine learning." In *Proceedings of the 2017 ACM on Asia conference on computer and communications security,* 506–519. ACM.

"PyTorch." 2019. Accessed June 1. `https://pytorch.org/`.

"Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3." 2019. Accessed June 1. https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.

Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. "You only look once: Unified, real-time object detection." In *Proceedings of the IEEE conference on computer vision and pattern recognition,* 779–788.

Redmon, Joseph, and Ali Farhadi. 2017. "YOLO9000: better, faster, stronger." In *Proceedings of the IEEE conference on computer vision and pattern recognition,* 7263–7271.

———. 2018. "Yolov3: An incremental improvement." *arXiv preprint arXiv:1804.02767*.

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. 2015. "Faster r-cnn: Towards real-time object detection with region proposal networks." In *Advances in neural information processing systems,* 91–99.

"Specific format of annotation Issue 60 AlexeyAB/Yolo_mark." 2019. Accessed June 1. https://github.com/AlexeyAB/Yolo_mark/issues/60.

Su, Jiawei, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. "One pixel attack for fooling deep neural networks." *IEEE Transactions on Evolutionary Computation.*

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition,* 1–9.

"tensorflow/cleverhans: An adversarial example library for constructing attacks, building defenses, and benchmarking both." 2019. Accessed May 14. https://github.com/tensorflow/cleverhans.

Thys, Simen, Wiebe Van Ranst, and Toon Goedeme. 2019. "Fooling automated surveillance cameras: adversarial patches to attack person detection." *arXiv preprint arXiv:1904.08653*.

Xia, Gui-Song, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. 2018. "DOTA: A large-scale dataset for object detection in aerial images." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 3974–3983.

"YOLO: Real-Time Object Detection." 2019. Accessed June 1. https://pjreddie.com/darknet/yolo/.