DPatch: An Adversarial Patch Attack on Object Detectors

Xin Liu[1], **Huanrui Yang**[1], **Ziwei Liu**[2], **Linghao Song**[1], **Hai Li**[1], **Yiran Chen**[1]
[1]Duke University, [2]The Chinese University of Hong Kong
[1]{xin.liu4, huanrui.yang, linghao.song, hai.li, yiran.chen}@duke.edu, [2]zwliu.hust@gmail.com

Abstract

Object detectors have emerged as an indispensable module in modern computer vision systems. In this work, we propose DPatch– a black-box adversarial-patch-based attack towards mainstream object detectors (*i.e.* Faster R-CNN and YOLO). Unlike the original adversarial patch that only manipulates image-level classifier, our DPatch simultaneously attacks the bounding box regression and object classification so as to disable their predictions. Compared to prior works, DPatch has several appealing properties: (1) DPatch can perform both untargeted and targeted effective attacks, degrading the mAP of Faster R-CNN and YOLO from 75.10% and 65.7% down to below 1%, respectively; (2) DPatch is small in size and its attacking effect is location-independent, making it very practical to implement real-world attacks; (3) DPatch demonstrates great transferability among different detectors as well as training datasets. For example, DPatch that is trained on Faster R-CNN can effectively attack YOLO, and vice versa. Extensive evaluations imply that DPatch can perform effective attacks under black-box setup, i.e., even without the knowledge of the attacked network's architectures and parameters. Successful realization of DPatch also illustrates the intrinsic vulnerability of the modern detector architectures to such patch-based adversarial attacks.

Introduction

As deep learning systems achieve excellent performance in many cognitive applications, their security and robustness issues are also raised as important concerns recently. Among them, object detector is an indispensable module in modern computer vision systems, and widely deployed in surveillance (Liu et al. 2015) systems and autonomous vehicles (Girshick 2015). It becomes an increasing vital task to comprehensively study their vulnerability to adversarial attacks (Papernot et al. 2017; Liao et al. 2017).

Many existing adversarial attacks focus on learning full-image additive noise, where the predictions of deep learning system are manipulated by injecting small perturbations into the input samples. Though the injected noises are small and sometimes even invisible to human eyes, these methods need to manipulate the whole image and become less impractical for the real-world physical attacks.

Recently, adversarial patch (Brown et al. 2017) is introduced as an practical approach of real-world attacks. The adversarial patch misleads a CNN classifier to predict any object that coupled with the pasted patch to a targeted class, regardless of the object's scale, position, and direction. The effectiveness of such adversarial attacks has been also proven in a black-box setup where the structure and parameters of the attacked neural networks are unknown to the attacker.

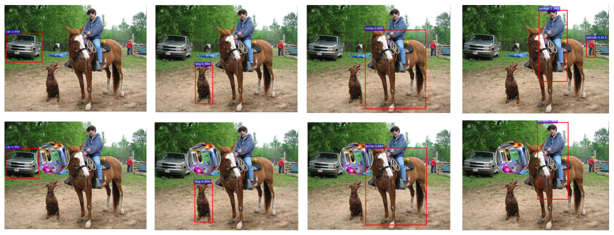


Figure 1: The original adversarial patch fails to attack object detectors. The first row is the original image. Faster R-CNN can detect multiple objects in the scene with a high accuracy. The second row is the image embedded with the Google's adversarial patch whose targeted class is $toaster$. Faster R-CNN is not influenced by the patch and can still be able to correctly recognize all the objects.

However, we found that the original adversarial patch technique is not able to fool object detectors such as Faster R-CNN (Ren et al. 2015) and YOLO (Redmon and Farhadi 2016), as shown in Fig.1. The reason resides at the detector architectures: Modern object detectors first locate the objects with different sizes at different locations on the image and then perform classification. Hence, the number of targets that need to be attacked in this case is much larger than that in a pure classification application (Avishek and Parham 2018). For example, Faster R-CNN generates approximately 20k regions for classification, which is far beyond an original adversarial patch can effectively attack.

Our key insight is that both bounding box regression and object classification need to be simultaneously attacked. Based on this observation, we propose DPatch – an iteratively trained adversarial patch that effectively attacks mainstream object detectors. A small DPatch, e.g., a 40-by-40 pixel region, can perform both untargeted attack and targeted attack: In an untargeted attack, the object detector cannot locate the correct region containing normal object; in a targeted attack, the object detector can only detect the DPatch but ignore any other objects on the image.

Extensive evaluations imply that DPATCH can perform effective attacks under black-box setup, i.e., even without the knowledge of the attacked network's architectures and parameters. Successful realization of DPATCH also illustrates the intrinsic vulnerability of the modern detector architectures to such patch-based adversarial attacks.

Related Work

Attacking Deep Learning Systems. Convolutional neural networks (CNNs) have been proven vulnerable to so called adversarial attack, where the classification result of the CNN can be manipulated by adding small perturbations onto its input examples (Christian et al. 2013; Eykholt et al. 2018; Xiao et al. 2018). The adversarial examples can be crafted by gradient based algorithms such as Fast Gradient Sign Method (FGSM) (Goodfellow et al. 2014) and Projected Gradient Descent (PGD) (Madry et al. 2017) or iterative methods such as DeepFool (Moosavi-Dezfooli, Fawzi, and Frossard 2016) and Carlini-Wagner (CW) attack (Carlini and Wagner 2016). Based on the concept of adversarial attack, an "adversarial glasses" (Sharif et al. 2016) was designed to fool a face recognition system so that the system recognizes the wearer as someone else. Most of the previous stuides on adversarial attack are about pixel-wise additive noise (Goodfellow et al. 2014; Moosavi-Dezfooli, Fawzi, and Frossard 2016; Carlini and Wagner 2016; Xiao et al. 2018). However, these attacks change all the pixels of the input image by a small amount, which tends to be not feasible in real-world vision systems such as web cameras and autonomous vehicles.

Adversarial Patch. To achieve a universal attack on real-world vision system, Google (Brown et al. 2017) recently designed a universal, robust adversarial patch that can be applied in real physical scene, causing a classifier to output any targeted class (Brown et al. 2017). Based on the principle that object detection networks are used to detecting the most "salient" object in an image, the adversarial patch can be trained to be more "salient" than other objects in the scene. Specifically, the patch is trained to optimize the expected probability of a target class (Brown et al. 2017). During the training process, the patch is applied to the scene at a random position with a random scale and rotation. This process enables the patch to be robust against shifting, scaling and rotating. Finally, the object in the scene will be detected as a targeted class of the patch.

Proposed Approach

Revisit Modern Detectors

Faster R-CNN. As mentioned before, Faster R-CNN is a typical two-stage detector. The first stage is to propose regions through a deep fully convolutional network, while the second stage is the Fast R-CNN detector that uses the proposed regions.

- **Region Proposal Network** After extracting features by convolutional layers, Faster R-CNN uses Region Proposal Network(RPN) to generate the region proposals that the object probably lies in. This Region Proposal Network takes as input an $n \times n$ spatial window (3×3 window as default) to slide over the last shared convolutional layer. These proposed regions generated by RPN will be mapped to the previous feature map for object classification later on.
- **Anchors Generation and Region of Interest** Faster R-CNN uses a 3×3 window to slide over the last shared convolutional feature map. An anchor is centered at the sliding window. It generates 9 anchor boxes at each sliding position associated with 3 scales and 3 aspect ratios. These anchor boxes are considered as the potential proposal. When evaluating, Faster R-CNN uses a *cls* layer, which outputs *2k* scores to estimate probability of object or not for each potential proposal. If the cls score is comparatively high, the potential proposal will be considered as the Region of Interest(RoI) where an object exist. Otherwise the proposal will be ignored.

  In order to attack the 2-stage detector like Faster R-CNN, it is necessary to invalidate the generation of correct region proposal. Our purpose is to make the region where the DPATCH exists as the only valid RoI, while other potential proposal should be considered not to own an object and thus, ignored.

YOLO. YOLO is introduced as a unified model for object detection. It reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. YOLO has great advantages of fast detection and high mAP.

- **Unified Detection** Unlike Faster R-CNN, YOLO is a single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. Specifically, it divides the input image into multiple grids, predicts bounding boxes and confidence scores for each grid. These confident scores reflect the confidence of the box containing an object, as well as the accuracy of the model predicting that box. This one-stage detection system trains on full images and directly optimizes detection performances (Joseph et al. 2015). By such optimized design, YOLO can run at 45 frames per second on a Titan X GPU and also achieve more than twice the mean average precision (mAP) of other real-time detection systems.
- **Bounding Boxes Prediction and Confidence Scores** Each grid in the image predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident that the box contains an object and how accurate the box is. If the confidence score is comparatively lower, that bounding box predicted by the grid will not be considered to contain a real object. Similarly, the grids where the DPATCH exists should be considered to have an object when attacking YOLO, while other grids should be ignored. That is, the grid containing a DPATCH has higher confidence score than others with normal objects.

DPATCH Formulation

Original Adversarial Patch. As introduced in the Introduction section, the adversarial patch proposed by

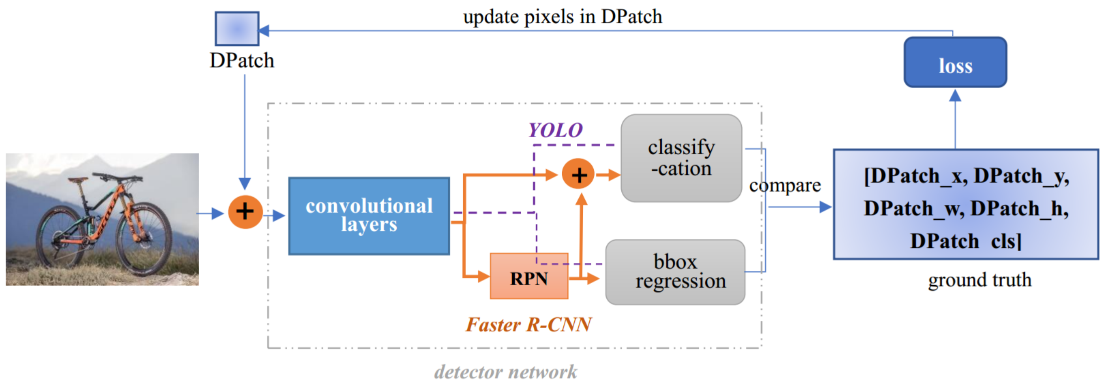


Figure 2: DPATCH training system: we add a randomly-initialized DPATCH to the image, utilize the detector network to do classification and bounding box regression based on the ground truth **[DPATCH_x, DPATCH_y, DPATCH_w, DPATCH_h, target_label]**. During back-propagation, we update the pixels of DPATCH.

Google (Brown et al. 2017) is aiming for maximizing the loss of a CNN classifier when the patch is applied to an input image. To make the patch effective on all the inputs and under potential transformations in the physical world, the patch is optimized over the expectation over random input images, locations and transformations.

$$\hat{P} = arg\max_{P} \mathbb{E}_{x,t,l}[logPr(\hat{y}|A(P,x,l,t))] \quad (1)$$

Equation (1) gives a formal description of the training objective of adversarial patch (Brown et al. 2017), where $A(P,x,l,t)$ denotes the input image produced by applying patch P to the original image x at location l with transformation t. Potential transformation includes scaling and rotating the patch. $Pr(\hat{y}|A)$ is the probability of classifying input A into the true label $\hat{y}$ providing the CNN classifier. Optimizing this objective could produce a patch that can effectively attack the classifier invariant of shifting, scaling and rotating when applied to any input.

Adversarial Patch on Object Detectors. Inspired by the Google adversarial patch, we propose DPATCH, an adversarial patch that works against state-of-the-art object detectors. We design DPATCH for both untargeted and targeted attack scenarios. For training the untargeted DPATCH, we would like to find a patch pattern $\hat{P}_u$ that maximize the loss of the object detector to the *true* class label $\hat{y}$ and bounding box label $\hat{B}$ when the patch is applied to the input scene x using "apply" function A, as shown in equation (2). And in a targeted attack setting, we would like to find a patch pattern $\hat{P}_t$ that minimize the loss to the *targeted* class label y_t and bounding box label B_t, as shown in equation (3).

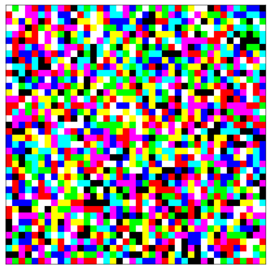
$$\hat{P}_u = arg\max_{P} \mathbb{E}_{x,s}[L(A(x,s,P);\hat{y},\hat{B})] \quad (2)$$

$$\hat{P}_t = arg\min_{P} \mathbb{E}_{x,s}[L(A(x,s,P);y_t,B_t)] \quad (3)$$

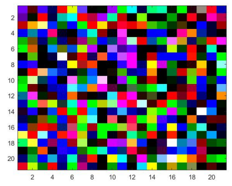
The "apply" function $A(x,s,P)$ means adding patch P onto input scene x with shift s. Here we uniformly sample the shift s within the scene during the training to make our patch shift invariant.

Object Detector Attacking System

DPATCH Training. As shown in Fig.2, we add a randomly initialized DPATCH pattern into the input image before it enters the detector system, and define the ground truth as [DPATCH_x, DPATCH_y, DPATCH_w, DPATCH_h, target_label]. DPATCH_x and DPATCH_y represent the location of the DPATCH while DPATCH_w and DPATCH_h represent its width and height. The bounding box label is [DPATCH_x, DPATCH_y, DPATCH_w, DPATCH_h], similar to the ground truth of detector's bounding boxes. For untargeted attack, target_label is defined as 0; for targeted attack, target_label is equal to the label of targeted class. Note that the detector system can either be Faster R-CNN or YOLO. The basic DPATCH training process starts with a pretrained Faster R-CNN or YOLO using Pascal VOC 2007 dataset (see details in **Experiments**).



(a) a 40-by-40 untargeted DPATCH that aims to attack YOLO



(b) A 20-by-20 *tv* targeted DPATCH that attacks Faster R-CNN

Figure 3: two DPATCH es.

DPATCH Design. Our default DPATCH is a 40-by-40 square which is attached onto the top left corner of the image.

- **Randomly-located DPATCH** In order to analyze the influences of different locations and to make DPATCH shift invariant, we randomly shift the location of the DPATCH while keeping all the pixels in the DPATCH same. Specifically, we randomly initialize the value of shift s when preparing DPATCH at each iteration of training, but leaving the pixel points in it unmodified. In such a case, each image in the dataset will be attached with the same DPATCH but at different locations. The size of randomly-located DPATCH is still 40-by-40, the same as the default setting.

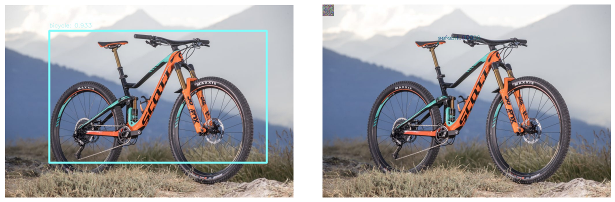

(a) No DPATCH (b) With DPATCH

Figure 4: YOLO cannot detect $bike$ after adding DPATCH Either the predicted bounding box or the classification result is incorrect. The predicted bounding box is just a dot, while the classification result is $sofa$ and $person$.

- **DPATCH of Different Targeted Classes** As there exist more than 10 object classes in the dataset used to train a detector, e.g., 20 classes in Pascal VOC 2007, it is intriguing to see whether mAP will fall down to a similar value if we set the DPATCH with different labels. For example, for targeted attack, we can set the ground truth as [0, 0, 40, 40, 15], 15 is the label index representing *person*. That is, we hope that the detector can only recognize the DPATCH and classified the DPATCH as *person*. In our experiment, we randomly select four classes, *bike*, *boat*, *cow* and *tv* from Pascal VOC 2007 and evaluate their attack effects.
- **DPATCH with Different Sizes** DPATCH size is another significant predetermined factor that could affect the effectiveness of DPATCH attack. There is a tradeoff between smaller patches that are harder to detect and defense, while larger patches that provide better attacking effect. In our experiment, we produce three different sizes of DPATCH es, namely, 20-by-20, 40-by-40 and 80-by-80 to test the efficiency of their attacks. In this way, we can better understand the relationship between DPATCH sizes and their attacking effects, so that we can find the minimal possible size of a patch for meaningful attack.

Transferability of DPATCH

Usually it is unknown that a detector is trained by which architecture. In such case, it makes more sense that the DPATCH trained by YOLO can also fool Faster R-CNN, or the DPATCH trained by Faster R-CNN with different classifiers can fool each other. To verify this idea, we train a DPATCH via YOLO and then add this trained DPATCH to input image, let Faster R-CNN work as the detector. If Faster R-CNN cannot correctly detect the object in the input image, our DPATCH attack should be considered as transferrable among detectors. Moreover, we train a DPATCHon COCO and then attack a Pascal VOC trained detector. If the mAP obviously decreased, DPATCHshould be considered as transferrable between datasets.

Experiments

Setup. We use the pretrained YOLO and Faster R-CNN with VGG16 and ResNet101 as basic networks. Pascal VOC 2007(Everingham et al. 2007) is utilized as the dataset, which includes 20 labelled object classes. For YOLO, the average mAP of the 20 classes is 65.7%; for Faster R-CNN with VGG16, the average mAP is 70.01%, for Faster R-CNN with ResNet101, the average mAP is 75.1%. Most previous work get similar detection mAP values as ours, so we can use these three networks as our pretrained detector networks. When performing targeted attack, we randomly choose one class from Pascal VOC 2007 to be our target class. Note that the default setting of YOLO is to generate 7×7 grids in each image, however, our experiment generate 10×10 grids instead. This change does not affect the overall detection performance of YOLO.

Untargeted DPATCH Attack

Fig.3(a) shows a 40-by-40 DPATCH that is designed as an untargeted adversarial patch. This DPATCH just aims to disable the detector(YOLO here), while it does not have a specific label. That is, it is unnecessary for the detector to accurately locate this DPATCH and to classify it as some class. As demonstrated, the original image Fig.3(b) where the bike should be correctly detected by YOLO, is detected wrongly as sofa and person in Fig.4(b).

After adding a untargeted DPATCH to each image in the testing dataset, mAPs of all classes are decreased obviously. As Table.1 and Table. 2 show, For YOLO, mAP is decreased from 65.7% to 0. For Faster R-CNN with ResNet101, mAP is decreased from 75.10% to 0.

Targeted DPATCH Attack

Fixed-sized and Fixed-located DPATCH. Fig.3(b) shows a 20-by-20 DPATCH, designed to attack Faster R-CNN with ResNet101, whose targeted class is tv. This tv DPATCH aims to make its occupied region to be recognized as the only valid RoI by Faster R-CNN. In Fig.5, our DPATCH covers the left top corner of each image, though the patch size(20-by-20) is small compared to other objects in the scene, it successfully fools the Faster R-CNN classifier and make it yield one result: the targeted class of the DPATCH. Hence, the function of multi-object detection and recognition of Faster R-CNN models is invalidated. The predicted probability is 0.997 of the first image of Fig.5, the predicted probability is 1.000 of the other three images. These predictions make sense because only the DPATCH occupied region can be recognized, other regions are ignored.

Specifically, the main purpose of applying such DPATCH is to make mean Accuracy Precision (mAP) of all classes in the dataset drop down to a lower value. The more mAP decreases, more successful the DPATCH attack is.

Table.1 demonstrates that after approximately 200k training iterations, this DPATCH could fool almost all the 20 classes in Pascal VOC 2007. The mAP falls down from 75.10% to 0.98%. We notice that at the start of training period (when training iteration is less than 40k), the falling speed of mAP is largest (see Fig.6). As the DPATCH accepts deeper training, its attack effect will gradually be weakened. Therefore, we can conclude that there exists a saturated point for training DPATCH. After that saturated point, increasing training iterations will no longer improve the attack effects. For tv, the saturate point of training iterations is about 180k.

Fig.7(a) shows another DPATCH attacking Faster R-CNN whose targeted class is $bike$. When we apply this DPATCH onto an image, all the bounding boxes determined by the

Table 1: Results on Pascal VOC 2007 test set with Fast R-CNN and ResNet101 when applying DPatch of different types

Faster R-CNN	plane	bike	bird	boat	bottle	bus	car	cat	chair	cow	table
no DPatch	74.80	80.20	77.60	64.50	61.50	81.10	86.70	86.40	55.70	89.30	69.60
untargeted DPatch	0.10	3.20	4.30	0.00	5.40	0.00	9.80	0.00	11.20	10.60	5.20
targeted DPatch	0.02	0.00	0.00	0.00	0.00	0.53	0.08	0.61	0.00	0.02	0.00
YOLO trained DPatch	2.27	**0.51**	**0.87**	2.27	**0.78**	1.52	4.55	0.62	1.17	3.03	2.10

	dog	horse	motor	person	plant	sheep	sofa	train	tv	mAP
no DPatch	87.40	84.50	80.00	78.60	47.70	76.00	74.60	76.60	73.70	75.10
untargeted DPatch	0.30	0.59	0.00	1.69	0.00	4.68	0.00	0.00	1.00	**2.90**
targeted DPatch	9.09	0.16	0.00	9.09	0.16	0.00	9.09	0.00	0.00	**0.98**
YOLO trained DPatch	2.02	3.37	1.30	0.94	0.53	0.43	3.03	1.52	1.52	**1.72**

Table 2: Results on Pascal VOC 2007 test set with YOLO when applying DPatch of different types

YOLO	plane	bike	bird	boat	bottle	bus	car	cat	chair	cow	table
no DPatch	69.50	75.60	64.00	52.30	35.60	73.40	74.00	79.60	42.10	66.10	66.90
untargeted DPatch	0.00	1.50	9.10	1.30	9.10	0.00	9.10	0.00	9.10	9.10	0.40
targeted DPatch	0.00	4.55	9.09	0.00	0.09	0.00	9.09	1.82	0.01	0.00	0.36
Faster R-CNN trained DPatch	0.01	**0.00**	**0.23**	0.02	**0.00**	0.00	0.00	0.00	0.01	0.00	0.00

	dog	horse	motor	person	plant	sheep	sofa	train	tv	mAP
no DPatch	78.10	80.10	78.20	65.90	41.70	62.00	67.60	77.60	63.10	65.70
untargeted DPatch	0.00	0.00	0.00	0.00	9.10	9.10	0.00	0.00	1.00	**0.00**
targeted DPatch	0.01	0.00	0.00	1.73	0.00	0.00	1.07	0.00	9.09	**1.85**
Faster R-CNN trained DPatch	0.00	0.03	0.00	0.07	0.00	0.00	0.00	0.00	0.01	**0.02**

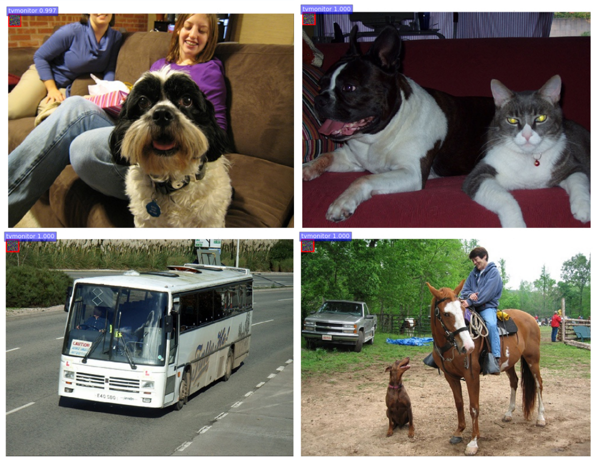


Figure 5: The DPatch is placed on the **left top corner** of the images. Faster R-CNN networks can on longer recognize other objects. Here, we apply a DPatch whose targeted class is tv, it can fully fool the Faster R-CNN networks though the DPatch size is comparatively small compared to other objects in the scene.

region proposals are disturbed. We observe that during the 200k training iterations of this $bike$ DPatch, it takes about 100k training iterations for the class loss to decrease to approximately 0, while the bounding box loss keeps comparatively higher after 200k training iterations. That is, the detector cannot locate this DPatch accurately, while its classification result is the targeted label of DPatch. Since our purpose is to disable the object detector, considering the detector(Faster R-CNN here) has already been attacked to give wrong classification result and also the bounding box of normal objects, this targeted DPatch attack should be considered successful. It is unnecessary to continue training this DPatch to make the detector locate it accurately.

Fig.7(b) shows a DPatch attacking YOLO whose targeted class is $person$. After we stick it to the image, the detector(YOLO here) can only give the classification as person, while the horses in the image cannot be detected.

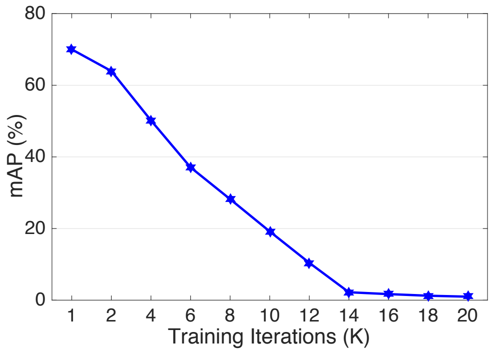
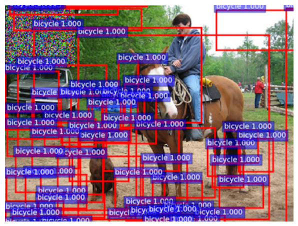
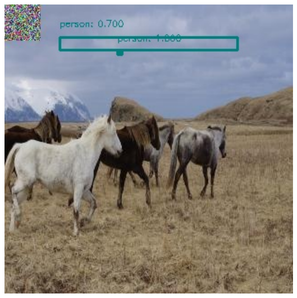


Figure 6: As training iterations accumulate, the falling speed of mAP gradually slow down, meaning the attack effects of DPatch will saturate at a point. For tv, the saturate point is approximately 200k training iterations.

Fixed-sized and Randomly-located DPatch. After experimenting with the fixed-located DPatch, our next trial is to randomly shift the DPatch in a scene, which means the same DPatch could appear in any location of the original image. The main purpose of such practice is to evaluate the attacking efficiency of different locations where the patch is placed. If the attacking efficiency, such as spending same training iterations to get similar mAP, does not differ from each other, it is unnecessary to design a specific attack region, which means attackers can place the DPatch in any area. We set the targeted class as $bike$, and kept the patch size as 20-by-20, and the attacked detector is Faster R-CNN.

Table.3 demonstrates the decreased mAP when Faster R-CNN is attacked by randomly-located and fixed-located DPatch. It is notable that randomly-located one does not improve the attack result. That is, recognition accuracy(mAP) of all classes have declined to a similar value no matter where the patch is located. This result makes


(a) targeted DPATCH attacking Faster R-CNN



(b) targeted DPATCH attacking YOLO

Figure 7: Successful DPATCH attack despite of inaccurate bounding boxes: The left is a $bike$ targeted DPATCH attacking Faster R-CNN, the right is a $person$ targeted DPATCH attacking YOLO. These wired bounding boxes are due to the inconvergence of the bounding box loss. Since our purpose is to disable the object detector but not to locate the DPATCH accurately, this DPATCH attack should be considered successful.

sense because Faster R-CNN detector will first extract thousands of region proposals all over the image, instead of finding a specific area. In this case, if the detector search the whole image in the first step, the region with DPATCH will be treated as the only valid RoI. After this detection process, all the detected objects will be misclassified as the targeted class, which also has no relation with the location of DPATCH. Similarly, YOLO also firstly sees the entire image, the location of the DPATCH will not influence the attack results. Therefore, we could place the DPATCH in an image without intentionally designing its location, which intensifies the feasibility of the attack.

Multiple-Sized DPATCH. Since all previous targeted attacks set the DPATCH size to be 20-by-20, we would prefer to observe the impacts of DPATCH size on detectors. We add two more sizes in this test: 40-by-40 and 80-by-80. It is expected that larger-size patch can decrease mAP to a lower value. Table.4 validates such expectation.

In order to avoid the influence of training iterations, we train these three DPATCH es for 200k iterations and make them approach saturated points. We observe that the smallest size for valid attack differ from individual classes. For example, 20-by-20 sized DPATCH is robust enough to attack $bike$, $bird$, $boat$ and so on, while 80-by-80 sized one still cannot thoroughly misclassify $bottle$, $motor$, $person$ and $plant$. Therefore, we can set the DPATCH size according to the classes we mainly want to attack.

Further Analysis

Training with Different Targeted Classes. It has been found that the bounding box loss is easier to converge for some class, e.g. tv, however, harder to converge for some other targeted classes like $bike$ and $person$. Therefore, we would like to explore whether DPATCH of different targeted classes can cause mAP drop down to a similar value after same training iterations.

We casually select two more classes to attack the same detector, Faster R-CNN: $boat$, cow to compare with tv and $bike$. Fig.8 shows that after 200k training iterations, these targeted classes cause mAP fall down to different levels. cow and tv both decline mAP to almost 0, while $bike$ and $boat$ shrink mAP to 24.72% and 33.50%.

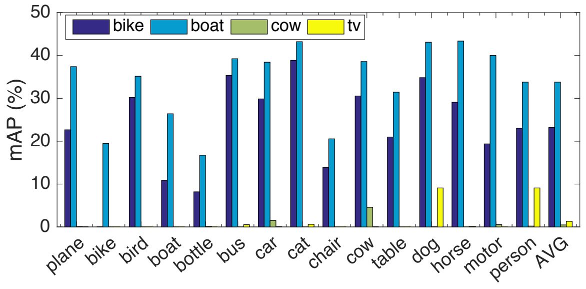


Figure 8: Per class mAP after attacked by DPATCH es of different targeted classes. $bike$ and $boat$ shrink mAP to 24.72% and 33.50%, while cow and tv shrink mAP to 0.38% and 0.98%. Therefore, the attack results differ from targeted classes.

Based on this finding, we can conclude that tv and cow are more efficient to attack Faster R-CNN networks. It could be better to set the targeted class as tv or cow rather than $boat$ or $bike$. Hence, we can select the most efficient targeted class to train the DPATCH if the dataset is known.

Detector Transferability of DPATCH. In the previous experiments, we attack the same detector as the DPATCH is trained on. However, it is more effective if the DPATCH is transferable among different detectors, specifically, we train the DPATCH via YOLO and it can also successfully attack Faster R-CNN. Fig.9 shows that a DPATCH trained by YOLO, whose targeted class is $person$, can fool Faster R-CNN. We apply this YOLO-trained DPATCH to each image in the testing dataset of Pascal VOC 2007 and make Faster R-CNN work as the detector, Table.1 demonstrates that mAP of some classes, e.g. $bike$, $bird$, $bottle$, are even declined to a lower value than attack on YOLO itself, while the attack on some other classes are less effective, e.g., $plane$, $boat$. However, mAP of all classes have considerable decrease. Similarly, Faster R-CNN trained DPATCH is also able to attack YOLO, see Table.2. In such case, we can say that our DPATCH can be designed as a universal and black-box adversarial attack for object detectors.

Dataset Transferability of DPATCH. To investigate the transferability of DPATCH among different datasets, we train an untargeted DPatch using COCO and test it on Pascal VOC trained detector. It is obvious that mAP decreases a lot after attaching the DPatch, shown in Table.5. Although the attack performance is not as good as the DPatch trained by the same dataset, DPATCH can still considerably influence the overall performance.

Why DPATCH Works. As explained before, our main purpose is to train a DPATCH pattern that once attached to the input image, the RoIs extracted by the detector is the region that DPATCH occupies. After extracting RoIs, detector performs classification tasks and bounding box regressions on these RoI areas while ignore others. In such case, if the DPATCH attacks the detector successfully, most ex-

Table 3: Results on Pascal VOC 2007 test set of fixed-located DPatch and randomly-located DPatch, whose targeted class is *bike* that attacks faster R-CNN. After 200k training iterations, mAP of Pascal VOC 2007 both decreased from 75.10% to around 25%. Therefore, the location of DPatch does not influence the attack results.

	plane	bike	bird	boat	bottle	bus	car	cat	chair	cow	table
fixed-located	25.66	0.03	32.48	11.03	8.68	35.36	30.19	38.97	13.79	30.55	21.7
randomly-located	25.69	0.03	32.16	11.54	9.68	36.01	30.12	38.66	14.09	30.13	22.00
	dog	horse	motor	person	plant	sheep	sofa	train	tv	mAP	
	35.77	29.13	19.44	23.02	9.33	33.61	36.28	41.09	32.16	**25.14**	
	36.01	29.01	19.39	23.17	9.33	33.58	36.30	41.11	32.19	**25.51**	

Table 4: Results on Pascal VOC 2007 test set after attacked by 20-by-20-sized, 40-by-40-sized and 80-by-80-sized DPatch . The targeted class are all *cow* that attacks Faster R-CNN. After same training iterations (200k in this case), larger sized adversarial patch can decrease mAP to a lower value.

	plane	bike	bird	boat	bottle	bus	car	cat	chair	cow	table
20-by-20	0.1	0	0	0	0.17	0	1.5	0	0	4.57	0
40-by-40	0	0	0.01	0	0.16	0	1.14	0	0	4.55	0
80-by-80	0	0	0	0	0.09	0	1	0	0	3.92	0
	dog	horse	motor	person	plant	sheep	sofa	train	tv	mAP	
	0.04	0	0.54	0.21	0.4	0.05	0.06	0.03	0	**0.38**	
	0.02	0	0.51	0.12	0.61	0.02	0	0	0	**0.36**	
	0	0	0.43	0.09	0.33	0	0	0	0	**0.29**	

Table 5: Transferability of DPatch among different detectors and datasets. The DPatch trained with COCO is able to attack the Faster R-CNN trained by Pascal VOC.

	Faster R-CNN+VOC	YOLO+VOC
no DPatch	75.10	65.70
COCO-trained DPatch	28.00	24.34
VOC-trained DPatch	2.90	0.00

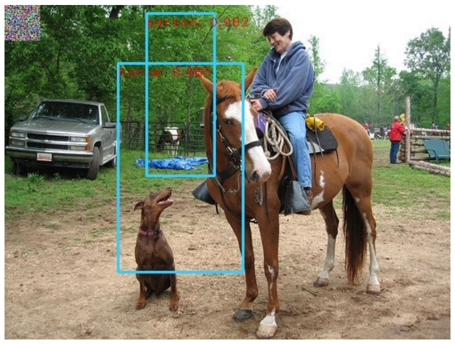
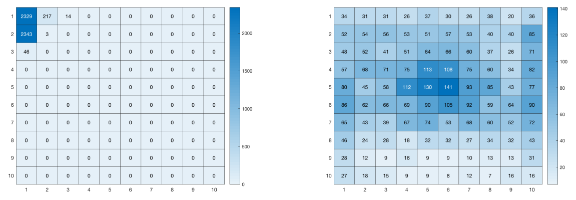


Figure 9: YOLO-trained DPatch successfully disables Faster R-CNN.

tracted RoIs should gather in the region where we attach DPatch. To verify this inference, we sum up the RoIs to find the most frequent area that RoI appears. As shown in Fig.10, the number in each grid represents the frequency that RoI occupies it. We attach a DPatch in the left upper corner of each image in the testing dataset, and let YOLO work as the detector. Fig.10(a) verifies that YOLO-extracted RoIs are concentrated on grid 1 and grid 2, where our DPatch locates. On contrary, Fig.10(b) shows that without DPatch, YOLO-extracted RoIs are distributed over all grids.

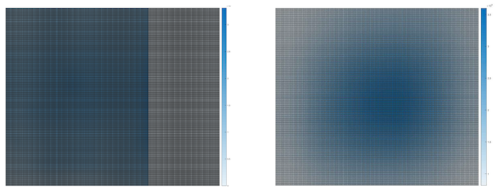
Similarly, Fig.11 demonstrates the distribution of RoIs extracted by Faster R-CNN with and without DPatch. It is clear that the regions that DPatch occupies are considered as RoIs more frequently, as shown in Fig.11(a). Instead,



(a) YOLO-extracted RoIs with DPatch

(b) YOLO-extracted RoIs without DPatch

Figure 10: The left is YOLO-extracted RoIs after adding DPatch in the left upper corner of each image. The number in each grid is the frequency of the RoI appears in it. Obviously, RoIs gather in the area where DPatch locates. On contrary, the right is extracted RoIs without DPatch. RoIs are distributed over all of the grids.



(a) Faster R-CNN extracted RoIs with DPatch

(b) Faster R-CNN extracted RoIs without DPatch

Figure 11: Darker the color, higher the frequency of RoI being predicted here. The left is Faster R-CNN extracted RoIs after adding DPatch, obviously concentrated on the left, while the right is Faster R-CNN extracted RoIs without DPatch, more evenly distributed.

RoIs distribute much more evenly without the influence of DPATCH, shown in Fig.11(b).

Such analysis verifies that our DPATCH performs a successful attack towards object detectors, mainly because all the RoIs is actually occupied by the DPATCH , instead of normal objects.

Conclusions

In this work, we successfully attack modern object detectors using our proposed DPATCH, where the adversarial attack is performed by learning and embedding a small patch in the input image. Compared to prior works, our DPATCH has several appealing properties: (1) DPATCH can perform effective attacks against mainstream modern detector architectures, such as the two-stage detector Faster R-CNN and the one-stage detector YOLO; (2) DPATCH is small in size and its attacking effect is location-independent, making it very practical to implement real-world attacks; (3) DPATCH demonstrates great transferability between different detector architectures as well as training datasets. For example, DPATCH that is trained on Faster R-CNN can effectively attack YOLO, and vice versa. Our experiments imply that DPATCH can perform effective attacks under black-box setup, i.e., even without the knowledge of the attacked network's architectures and parameters. The successful realization of DPATCH also illustrates the intrinsic vulnerability of the modern detector architectures to such patch-based adversarial attacks.

References

Avishek, J., and Parham, A. 2018. Adversarial attacks on face detectors using neural net based constrained optimization. In *Computer Vision and Pattern Recognition(CVPR)*.

Brown, T. B.; Man, D.; Roy, A.; Abadi, M.; and Gilmer, J. 2017. Adversarial patch. *arXiv preprint arXiv:1712.09665*.

Carlini, N., and Wagner, D. 2016. Towards evaluating the robustness of neural networks. *arXiv preprint arXiv:1608.04644*.

Christian, S.; Wojciech, Z.; Ilya, S.; Joan, B.; Dumitru, E.; Ian, G.; and Rob, F. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Everingham, M.; Van Gool, L.; Williams, C. K. I.; Winn, J.; and Zisserman, A. 2007. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

Eykholt, K.; Evtimov, I.; Fernandes, E.; Li, B.; Rahmati, A.; and Xiao, C. 2018. Robust physical-world attacks on deep learning visual classification. In *Computer Vision and Pattern Recognition(CVPR)*.

Girshick, R. 2015. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*.

Goodfellow; I.J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Joseph, R.; Santosh, D.; Ross, G.; and Ali, F. 2015. You only look once: Unified, real-time object detection. In *Computer Vision and Pattern Recognition(CVPR)*.

Liao, F.; Liang, M.; Dong, Y.; Pang, T.; Hu, X.; and Zhu, J. 2017. Defense against adversarial attacks using high-level representation guided denoiser. *arXiv preprint arXiv:1712.02976*.

Liu, Z.; Luo, P.; Wang, X.; and Tang, X. 2015. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, 3730–3738.

Madry, A.; Makelov, A.; Schmidt, L.; and Tsipras, D. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Moosavi-Dezfooli, S.-M.; Fawzi, A.; and Frossard, P. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2574–2582.

Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z. B.; and Swami, A. 2017. Practical black-box attacks against machine learning. *2017 ACM Asia Conference on Computer and Communications Security*.

Redmon, J., and Farhadi, A. 2016. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*.

Sharif, M.; Bhagavatula, S.; Bauer, L.; and Reiter, M. 2016. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 1528–1540.

Xiao, C.; Zhu, J.-Y.; Li, B.; He, W.; Liu, M.; and Song, D. 2018. Spatially transformed adversarial examples. *arXiv preprint:arXiv:1801.02612*.