APPROVAL SHEET

Title of Dissertation: Universal Adversarial Patches

Name of Candidate: Koninika Patil
Master of Science, 2017

Dissertation and Abstract Approved: _____

Dr. Hamed Pirsiavash
Assistant Professor
Department of Computer Science
and Electrical Engineering

Date Approved: 11/28/17

NOTE: *The Approval Sheet with the original signature must accompany the thesis or dissertation. No terminal punctuation is to be used.

ABSTRACT

Title of Thesis: UNIVERSAL ADVERSARIAL PATCHES

Patil Koninika Balkrishna, M.S. Computer Science, December 2017

Thesis directed by: Dr. Hamed Pirsiavash, Professor
Department of Computer Science and
Electrical Engineering

Deep learning algorithms have gained a lot of popularity in recent years due to their state-of-the-art results in computer vision applications. Despite their success, studies have shown that neural networks are vulnerable to attacks via perturbations in input images in various forms, called adversarial examples. Adversarial examples pose a severe security threat because they expose a flaw in machine learning systems. In this thesis, we propose a method to generate image-agnostic universal adversarial patches for attacking image classification and object detection using latent contextual information. Our experiments show that for classification, replacing a small part of an image with a universal adversarial patch can cause misclassification of more than 40% images. In object detection, we attack each category of objects individually and the best patch causes approximately 20% images to be misclassified when attacking images of the bird category. We also demonstrate that photos taken of adversarial examples containing the adversarial patch on a cell-phone, can also fool the network. Thus, we show that adversarial examples exist in the physical world which can cause harm to AI-based systems.

Keywords: Adversarial examples, Image Classification, Convolutional Neural Networks, Object Detection

Universal Adversarial Patches

by

Patil Koninika Balkrishna

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
M.S. Computer Science
2017

Dedicated to my family and friends who made the journey worthwhile and my advisers and mentors who guided me to the destination

ACKNOWLEDGMENTS

I would like to thank my adviser and mentor Dr. Hamed Pirsiavash for guiding me through my research, and for encouraging me to pursue and tackle interesting problems. His keen insight and observations were crucial to the progress of my research. Also, I would like to thank Dr. Tim Oates and Dr. Tim Finin for agreeing to be on my thesis evaluation committee. I am fortunate to have had so many amazing friends throughout my graduate studies. I would like to thank all Shruti, Anusha, Abhijeet, Devendra, Kshipra, Rujuta, Prutha, Apeksha, Abhilash, Siraj, Siddharth, Tejas, Ketan, Amee and Julia for making my masters degree experience at UMBC memorable and worthwhile. Also, my gratitude to my colleagues, Rose, Ananth, Vipin, Rahul, Maryam, Erfan, Aniruddha, Akshay and Ojesh at the Computer Vision Lab whose cooperation and immense support made my journey smooth and successful.

The success of my work would not be possible without the support of my parents, who always believed in me, my sisters who taught me that hard work always pays off and to always have a smile on my face, and the rest of my family, who are a constant source of inspiration to me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Image understanding tasks such as image classification and object detection (Szegedy *et al.* 2013) are closing in on human level performance since 2013 due to the immense success of deep learning algorithms. However, a lot of recent research has shown that neural networks are vulnerable to attacks via small, almost imperceptible perturbations induced in the input. These perturbations of the input, known as adversarial examples, are deliberately generated to push the network to misclassify images with high confidence. The fact that such a small perturbation can change the network's output shows that neural networks learn very differently from how humans learn, and they are somehow unable to understand the underlying concepts in images the way we do.

Adversarial examples are like optical illusions for computers, and they pose a severe threat to AI-based systems. Adversarial examples can either be targeted, where the network is pushed towards an incorrect target class, or non-targeted where the network is pushed away from the correct class. One of the methods of generating adversarial examples is by solving an optimization problem, by maximizing the loss function rather than minimizing it. (Szegedy *et al.* 2013), (Goodfellow, Shlens, & Szegedy), (Moosavi-Dezfooli *et al.* 2016) show ways of generating adversarial

examples. (Kurakin *et al.* 2017) even demonstrate that it is possible to capture photos of adversarial examples. Taking cell-phone pictures of adversarial examples printed out on paper, and passing them as input to trained neural networks still results in these photos being misclassified. Therefore, adversarial examples exist in the digital world as well as the physical world.

FIG. 1.1. Example of a universal adversarial patch. The first image shows the original image which is classified as a *Panda*. The second image contains the adversarial patch in the corner, and is therefore classified as a *St. Bernard* by Alexnet. The third image shows the Universal Adversarial Patch which simply needs to be applied to an image to fool a network.

In this thesis, we propose a non-targeted approach for creating adversarial examples by generating an image-agnostic universal adversarial patch, which could potentially be used in real-world attacks. Earlier methods have concentrated on making the perturbation vector small in terms of the magnitude of difference between the original image and the adversarial image, and push the network boundaries by tweaking

the physical appearance of objects in the images. In contrast to earlier methods, we introduce a patch which is small in terms of size, in an image, as shown in figure 1.1. These patches are successful in fooling the image classifier, Alexnet (Krizhevsky, Sutskever, & Hinton) and to some extent an object detector, YOLOv2 (**?**). Our adversarial patch works because image classifiers and object detectors rely on context information for making accurate predictions. To explain Context with an example: a classroom contains a board, a podium, tables and chairs, but you wouldn't expect a bear to be present there. Therefore, if a bear is actually present in a classroom setting, a neural network may get confused by the apparent mismatch in context, and would probably not classify it as a bear, but as a human. Thus, using the patch, we aim to change the context of the image and confuse CNNs. The patch which we use occupies a small part of the image and it is non-additive, that is, it replaces the pixels of the original image where it is applied. It is small enough, such that humans can recognize all objects present in the image after the patch is applied. The patch is universal across all the input data and is, therefore, image agnostic, not network agnostic. As a practical application, if the patch could be printed out and stuck on a car or stop sign, it could cause a lot of harm to self-driving cars.

Thus, to summarize, the following are the main contributions of this thesis:

1. We propose a method for generating an image-agnostic universal adversarial patch which can fool state-of-the-art deep neural networks.

2. We show that we can learn patches to fool image classification, as well as, to some extent object detection.

3. With extensive analysis, we show that context plays a big part in image understanding tasks.

4. We show that cell-phone images of adversarial patches can also be used in the real-world to deceive computer vision algorithms.

The remainder of the thesis has been organized as follows. Chapter 2, Background and Related Work, discusses convolutional neural networks for classification and detection, and relevant research of adversarial examples. Chapter 3, Algorithm and Methodology, delves into the details of our experimental setup and our algorithm for generating adversarial patches. Chapter 4, Datasets, describes the datasets used for our experiments. In Chapter 5, Evaluation and Experimental Results, we evaluate the robustness of image classification and object detection to our patch. We conclude and discuss future work in Chapter 6, Conclusion.

Chapter 2

BACKGROUND AND RELATED WORK

Adversarial examples are trending in deep learning research, as they expose a flaw in machine learning models. There is a tremendous amount of research taking place to attack as well as defend deep networks. In this chapter, we talk briefly about convolutional neural networks for image classification and object detection; and relevant research carried out in adversarial examples.

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) came back to life after decades of existence when it was feasible to train them on a large number of images in comparatively less processing time, due to the availability of high-performance computing hardware and a large amount of data helped. In 2012, (CNNs) became popular after (Krizhevsky, Sutskever, & Hinton) used a CNN based model to win ILSVRC 2012 (ImageNet Large-Scale Visual Recognition Challenge). Since then, CNNs have diversified to tackle several visual understanding, natural language processing as well as problems in other deep learning areas.

Image classification has been immensely successful since the advent of Alexnet, VGG-16, VGG-19 (Simonyan & Zisserman 2015) and so on. Some models such as

Inception (**?**) and Resnet (He *et al.*) have achieved higher accuracy than human performance for classification. Object Detection techniques have steadily improved since (Girshick *et al.*) used CNNs for object detection. He proposed a method to generate region proposals in an image, compute CNN feature vectors for each of the regions and then classify the regions using a linear SVM. Faster R-CNN (**?**) goes a step further and produces region proposals using CNNs as well. YOLOv2 (You only look once), which is an improved version of YOLO (Redmon *et al.*), is a real-time object detection system which gives a mean average precision (mAP) of 78.6% on the 20 categories of the Pascal VOC dataset.

2.2 Adversarial Perturbations

Despite the impressive performance of CNNs, they fall prey to adversarial attacks. Adversarial examples were discovered when (Szegedy *et al.* 2013) was trying to understand how CNNs work by finding the transformation of an image belonging to one class into another class. He found instead, images which could fool the network into believing that it belonged to the target class by adding an undetectable amount of noise. Since then, many attacks and defenses for adversarial examples have been devised within a short span of time.

2.2.1 Attacks

(Goodfellow, Shlens, & Szegedy) in 2015 proposed a fast method for computing adversarial perturbations, Fast gradient-sign method (FGSM). FGSM defines an adversarial perturbation as the direction in image space which maximises the cost function by taking a step of gradient ascent in the direction of sign of gradient.

(Moosavi-Dezfooli *et al.* 2016) found a single universal adversarial perturbation

which can fool a large number of images while being imperceptible to humans. In the paper they have explained how the perturbation can generalize to different network architectures as well.

(Lu *et al.* 2017) demonstrated how classifiers and detectors are different and explained why standard object detectors aren't fooled by adversarial examples, whereas classifiers are easily fooled. They showed experimental results on YOLOv2 and Faster R-CNN, using printed stickers stuck on stop signs as well as adversarial noise hidden in graffiti and art. Their method is based on their earlier paper by (**?**).

2.2.2 Defences

While a lot of ways of generating adversarial examples have been found, the cause of their existence is still not certain. However, there are some researches dedicated to mitigating the effect of adversarial examples.

(Papernot *et al.*) used a methodology for defending against adversarial examples based on knowledge distillation by (Hinton, Vinyals, & Dean 2015) called Defensive distillation. They leverage the knowledge extracted from the output predictions of a CNN to increase the network's robustness to adversarial samples. However (Carlini & Wagner) show that defensive distillation is in fact, not more robust to adversarial examples.

Adversarial training is one of the strategies used for defending against adversarial examples, where the network is trained on original as well as adversarial images. (Kurakin *et al.*) apply adversarial training at a largescale on the ImageNet dataset. They show that adversarial training proves beneficial, especially in case of single-step adversarial attacks. (Tramèr *et al.*) extend the adversarial training idea to training an ensemble adversarial training method to increase the network's robustness to black-box attacks.

(Metzen *et al.*) and more recently Safetynet by (Lu, Issaranon, & Forsyth) devised a defence against adversarial examples which uses an adversary detector which classifies an image as adversarial or not adversarial by looking at the internal layers and representations of the network. If an image is classified as adversarial, it is rejected from further processing.

Chapter 3

ALGORITHM AND METHODOLOGY

The main idea behind our attack is to introduce a small patch in the image such that a classifier or detector will be blind to the attacked class in the image. We need to search for a perturbed patch which pushes the representation vector for each class of the image into a different class. Our search is an optimization problem where we optimize the patch to maximize the cost function instead of minimizing it. We find the universal perturbed patch by extending the Gradient step method provided by (Szegedy *et al.* 2013).

While training, we learn the patch by iterating over each image in the dataset and taking a step of gradient ascent for each image (or batch). While training, we also carry out validation on a small set of images taken from the testing dataset. During validation, we obtain the network prediction for original images and compare it with the prediction for adversarial images, and report the number of misclassified objects or the fooling rate. After training the patch for a sufficient number of iterations, we use the patch which reported the best fooling rate for the validation dataset.

While iterating over each image in the dataset, we perform the following steps:

1. Initialize the adversarial patch with random values within permissible limits of the image at the beginning of the first iteration.

2. Add the adversarial patch to the image by replacing the respective pixels in the image.

3. Find the direction where we minimize the likelihood of specified class or maximise the loss function. We find this by taking the loss gradient with respect to the image.

4. Crop the gradient to 30×30 pixels around the noise patch location

5. Update all parameters in the noise vector by taking a step of gradient ascent

6. Repeat this process on each image in the dataset for a sufficient number of iterations over the dataset

Thus, at each data point we are taking one step away from predicting the correct class. Also, we have to clip the intensity of the outlier patch pixels after each updation to contain them within the limits of RGB image. This procedure is shown in Fig. 2.

In our attack we can adjust three parameters: Learning rate, Batch size and Patch location. For both classification and detection, we train the patch on a pre-trained network and keep the network weights fixed, while adjusting the perturbed patch.

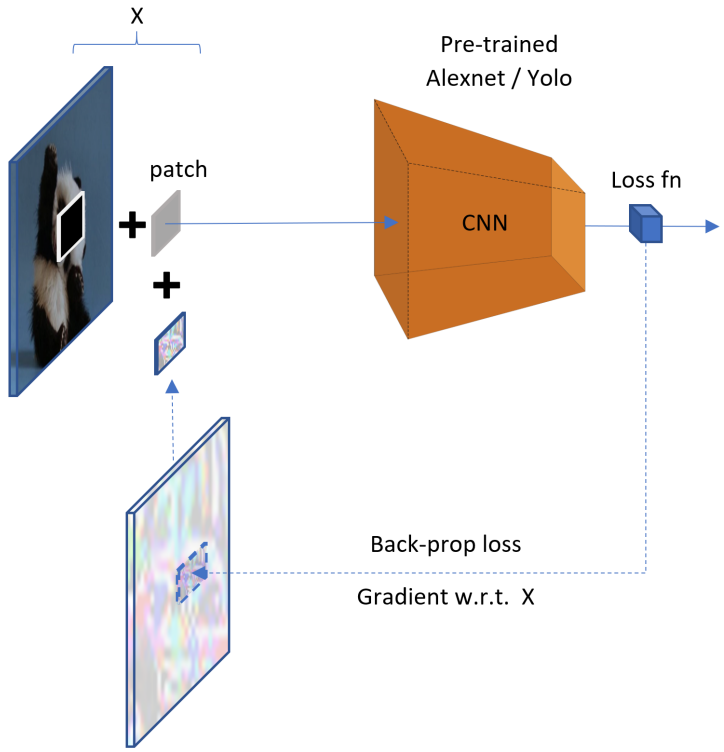


FIG. 3.1. Procedure for training the adversarial patch

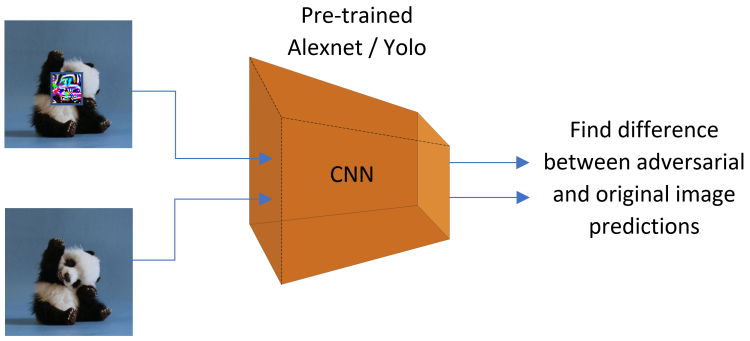


FIG. 3.2. Testing the adversarial patch against the original input

3.1 Classification

For our classification experiments, we use the Alexnet CNN model, which has ve convolutional layers, two fully connected layers and an output layer having nodes corresponding to the number of classes in the dataset. We use the Imagenet (ILSVRC 2012) dataset, which has 1000 categories. The 30×30 noise patch is initialised with random values before the first iteration and it is applied to each input image, thus creating an adversarial image. This 224×224 adversarial image is given as input to Alexnet and the output of forward propagation on the network is the prediction of the class of the adversarial image.

The loss function used in classification is cross-entropy loss. The gradient of cross-entropy loss is back propagated onto the input image and we thus we the perturbed patch. The Equation 3.1 represents the loss function for image classification:

$$loss = -\log\left(\frac{exp(x[class])}{\sum_j exp(x_j)}\right) \tag{3.1}$$

3.2 Detection

For attacking object detection, we use the YOLOv2 model whose architecture is based on GoogleNet (**?**) and has 24 convolutional layers. The output vector of this network contains predictions for each object detected in the image. The output contains offsets x, y, w, h where x and y are the center coordinates of the predicted object, and w, h are the width and height. The output also contains an 'objectness confidence' score, which denotes the probability that the predicted bounding box contains an object. The last output is a probability vector which provides the probabilities of the object belonging to each category in the dataset.

This probability vector is the output we are interested in. The loss function for YOLO consists of Mean Square Error losses for the offsets and Objectness confidence, and a Cross-entropy loss for classification. We are only concerned with the classification loss so, this is the loss whose gradient we back-propagate onto the image to update our patch. Just like classification, we add our randomly initialized patch to the input image and we find the class probabilities and cross-entropy loss for this adversarial image. We proceed to do a gradient ascent step with cross-entropy loss obtained. Cross-entropy loss can be defined as follows:

$$class\_loss = Prob\Big(class(x) == A\Big) = -\log\left(\frac{exp(x_A)}{\sum_j exp(x_j)}\right) \tag{3.2}$$

Equation 3.2 shows the Probability of the class of the image x being equal to A, the attacked or actual class. The summation over j shows the summation of probabilities of x belonging to all the classes in the dataset.

When we simply used the cross-entropy loss function, there is a possibility that the patch itself may be predicted as the attacked class. To suppress the patch from learning features of the attacked class, we introduce another cross-entropy loss which penalizes the patch for being predicted as the attacked class. This can be accomplished by adding the patch location to the ground truth vector for object detection, where the data is the patch itself, and the label is the attacked class. Thus, if the patch is predicted as the attacked class, it will be penalized according to our gradient ascent algorithm. The loss function for penalizing the patch can be written as follows:

$$patch\_loss = Prob\Big(class(patch) == A\Big) = -\log\left(\frac{exp(patch_A)}{\sum_j exp(patch_j)}\right) \tag{3.3}$$

Equation 3.3 shows the Probability of the class of the image x being equal to A, the attacked or actual class. The final loss that we use for object detection is the sum of these two losses 3.2 and 3.3 respectively in Equation 3.4.

$$loss = class\_loss + patch\_loss \tag{3.4}$$

3.3 Algorithm

Consider a model M which is the neural network we are trying to fool. $X = \{x_1, x_2, ..., x_m\}$ is the set of input images to M and $Y = \{y_1, y_2, ..., y_m\}$ is the set of ground truths for each X. Let p be the perturbed patch which replaces a part of the image x, and let us call this perturbed image x_p. Let m be a binary mask corresponding where the part of the image to which the patch is added is made 0. Thus, the image region has a value of 1 and the noise region has a value of 0. On element-wise multiplication of x with m and then adding p to the image, we obtain x_p. The patch is initialized with random values before the first iteration. The patch is bounded by the permissible limits of an image (0 to 1 or 0 to 255). The main idea is to use gradient ascent instead of gradient descent and train the patch p such that every step of gradient ascent takes the network away from predicting the correct output for the image. If J is the cost function for training M and W are the model M's weights, we can formalize the patch p for one image as:

- Replace a patch on the image with noise patch $(x_1)_{p_0} = x_1 \odot m + p_0$
- $p_1 = p_0 + \alpha \times \nabla_{(x_1)_{p_0}} J\left(W, (x_1)_{p_0}, y_1\right)$; p_0 is initialized to random permitted values for an RGB image

To make the patch universal, we train it on all the input images and keep accumulating the loss. Thus the patch for the i^{th} image will be:

$$p_i = p_{i-1} + \alpha \times \nabla_{(x_i)_{p_{i-1}}} J\left(W, (x_i)_{p_{i-1}}, y_i\right)$$

We evaluate our model based on the reduction in precision or the number of predictions changed from the predictions made on the original image.

Algorithm 1: calculate universal adversarial patch

input : set of images X, set of corresponding targets Y and pre-trained model M on the given images X, coordinates of patch C

output: universal adversarial patch p

Initialization: patch p is initialized to random permitted values for an RGB image.

for sufficient number of iterations over X **do**
  for image $x_i \in X$ with label $y_i \in Y$ **do**
    $(x_i)_{p_{i-1}} = x_i \odot m + p_{i-1}$
    $p_i = \alpha \times \nabla_{(x_i)_{p_{i-1}}} J\left(W, (x_i)_{p_{i-1}}, y_i\right)$
    $p_i =$(Crop p_i at C, coordinates of patch, to 30x30)
    $p_i = p_i + p_{i-1}$
    clamp p_i between 0 and 1
  end for
end for

Chapter 4

DATASETS

This chapter briefly describes the datasets we have used. For classification, we use the Imagenet ILSVRC 2012 (Deng *et al.*) dataset and its characteristics are described in section 4.1. For object detection, we use the Pascal VOC (Everingham *et al.* 2015) dataset to train and test our patch. We have described this dataset below in section 4.2.

4.1 ImageNet

The Imagenet dataset is a vast collection containing 14197122 images gathered from multiple sources such as Google, Yahoo, Flickr etc. These images have been annotated by humans using Amazon's crowd-sourcing platform for human intelligence tasks: Mechanical Turk. Imagenet has organized into a hierarchy based on Wordnet and has 1000 categories. We use the flat labels, without leveraging the hierarchy information. Imagenet is used every year for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates different object detection and image classification algorithms.

For our image classification experiments, we use approximately 1280000 images for training and 50000 images for testing.

4.2 Pascal VOC

The Pascal Visual Object Challenge (VOC) dataset was another challenge, to assess object recognition from various data sources. It was started in the year 2005. The object classes of Pascal dataset have been organised as into the following categories.

- **Person**: person
- **Animal**: bird, cat, cow, dog, horse, sheep
- **Vehicle**: aeroplane, bicycle, boat, bus, car, motorbike, train
- **Indoor**: bottle, chair, dining table, potted plant, sofa, tv/monitor

Pascal VOC dataset has been annotated for object detection, specifying localistion information for each object within the image. It provides the coordinates of the center of the image, height and width of objects which can be used for training object detection systems.

Since the number of images in the Pascal dataset is limited, we combine the training images from the years 2008-2012, as well as the validation images to form our training data. There is a separate dataset for testing. We are not explicitly provided the ground truth for the test images. We have to upload the predictions for test images on to the Pascal VOC challenge server to get the mean average precision results. This can be done by using the script provided by the organisers of the challenge.

Combining the training and validation sets, we get 23988 training images, and 7010 testing images from the test dataset. Since we attack each class in object detection separately, we train the patch to fool the class only on the images containing objects belonging to the attacked class. In table 4.1 given below, we enlist the training and testing images we have used according to each individual class.

task	**aero**	**bicycle**	**bird**	**boat**	**bottle**	**bus**	**car**	**cat**	**chair**	**cow**
training	908	795	1095	689	950	607	1874	1417	1564	444
testing	204	239	282	172	212	174	721	322	417	127
task	**table**	**dog**	**horse**	**mbike**	**person**	**plant**	**sheep**	**sofa**	**train**	**tv**
training	738	1707	769	771	6095	772	421	736	805	831
testing	190	418	274	222	2007	227	97	223	259	229

Table 4.1. Images by category in Pascal VOC Dataset

Chapter 5

EVALUATION AND EXPERIMENTAL RESULTS

In Chapter 3, we have described the algorithm for generating the universal adversarial patch. In this chapter, we use the algorithm to train patches for classification and detection, and show detailed results of the performance of classifiers and detectors after introducing the adversarial patch in the image at various locations.

5.1 Classification

We have used Alexnet as the classifier model for our experiments and measured its performance on a patch trained on the Imagenet dataset. As mentioned in chapter 4, we used 1281000 training images and 50000 validation images. We pretrain Alexnet on Imagenet to obtain a top-1 accuracy of 52.3% and top-5 accuracy of 76.81%. To train the patch for classification, we do only one iteration over the entire training dataset using the pre-trained Alexnet model. We tuned training parameters, such as batch size and learning rate. We also conducted experiments by placing the patch in different locations. As shown in the algorithm in chapter 3, we found the perturbed patch by back-propagating the gradient of loss with respect to the image. To evaluate the learned patch, we added the patch onto the image after masking, and report the

result of the ratio of images whose labels changed when comparing the predictions of the original image and the adversarial image. We carried out experiments by placing the patch in the center of the image, in the top left corner of the image and in random locations all over the image. After some experimentation, we settled on a learning rate = 0.1 and batch size = 16. The size of the input image is 224×224 and patch size is 30×30.

5.1.1 Placing the patch in the top-left corner

In this experiment, we train the patch by placing it in the top-left corner of the image. The x and y coordinates of the starting point (top-left corner) of the patch on the image in this experiment are (10, 10). The patch is initialized randomly within the permissible range for an RGB image. When we get the gradient of each image with respect to the adversarial image, we crop it to a 30×30 square at (10, 10) to get the gradients at the patch location and do gradient ascent by adding the gradient to our patch. The best fooling ratio that we achieved for the top-left corner patch was after seeing about 650000 images. With a patch of size 30×30, we have achieved a fooling rate of 42.1% when placing the patch in the top-left corner. When the same patch is placed randomly along the edge of the image at test time, we obtain a label change ratio of 27%. This shows that the patch is not sensitive to the location where it has been trained, and works well even when placed at a different location.

5.1.2 Placing the patch at a random location on the edge of the image

This experiment is exactly like the top-left corner experiment, except that we placed the patch randomly between the edge and bordering 20 pixels of the image. We obtain a label change ratio of 32.97% in this experiment. We found the best patch after seeing approximately 300000 images. We conduct this experiment to show that

the patch we have learned is robust to change in location and can be placed any where in the image.

5.1.3 Placing the patch at random location anywhere in the image

In this experiment too, we place the patch randomly, but the patch can be placed uniformaly all over the image. We obtained a label change ratio of 44.54% in this experiment. We found the best patch after seeing approximately 840000 images.

5.1.4 Placing the patch at the center of image

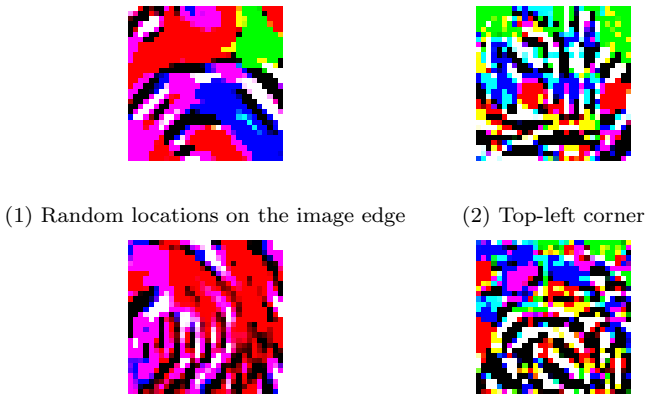
On training the patch by placing it in the center of the image, we obtain the best fooling ratio, i.e. 64.59%. However, in this scenario a portion of the object is covered by the patch, since Imagenet centers the objects. The best patch was after seeing approximately 520000 images.

Thus, in classification, we show that even though our patch is not overlapping on the object and is placed in a corner, it is capable of changing the class label of the image. This shows that somehow the context of the image is being changed in a way that the prediction of the network is pushed into a different class which fits the context better.

The results of the classification experiments are reported in table 5.1. The table specifies the patch location along with the ratio of labels changed for input images from the training dataset and testing dataset. We show some positive results on applying the random patch in figure 5.2 and some negative results in figure 5.3. The first picture shows the original image and the second picture is the adversarial image.

Classification Experiment	Training Data	Testing Data
Patch in the top left corner of the image	79.89	42.10
Patch located randomly on edge of the image	84.02	32.97
Patch located randomly in the image	76.56	44.54
Patch located at the center of the image	99.73	64.59

Table 5.1. Classification results: ratio of predicted labels changed after adding the learned patch on training and testing data



(1) Random locations on the image edge (2) Top-left corner

(3) Random locations on the image (4) Center of the image

FIG. 5.1. Adversarial patches learned for classification on the Imagenet dataset

(1) original: stingray; adversarial: doormat, welcome mat

(2) original: shower cap; adversarial: Chesapeake Bay retriever (dog breed, also UMBC Mascot)

(3) original: radiator; adversarial: paintbrush

(4) original: sliding door; adversarial: pay-phone, pay-station

FIG. 5.2. Positive results for adversarial patches learned for classification on the Imagenet dataset

(1) original: Arabian camel, dromedary, Camelus dromedarius ; adversarial: Great Dane

(2) original: vine snake; adversarial: tree frog, tree-frog)

FIG. 5.3. Negative results for adversarial patches learned for classification

5.2 Detection

As described in secton 3.2, we generated adversarial patch for YOLO object detection algorithm. We trained as well as measured its performance on the Pascal VOC dataset, consisting of 16000 training images and 5000 validation images. While training we do several iterations over the entire training dataset. We first pre-trained Yolo on the Pascal VOC training data, and then used this pre-trained model to learn our patch. The mAP each class and average precision achieved on the pre-trained Yolo model is given in table 5.2.

aero	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	table
76.82	78.38	72.59	63.41	48.05	78.49	79.35	87.03	51.45	78.72	72.1

dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
81.57	80.29	78.94	75.3	49.07	77.14	72.75	86.17	73.52	73.06

Table 5.2. mAP and AP achieved on pre-trained Yolo network on Pascal-VOC dataset

Object Detection is different from Classification. Firstly, the number of categories is far less than the classification dataset categories (20 as compared to 1000) and therefore the network may learn better representations of each class. Also, detection predicts a localization component as well as an objectness confidence score, which is the probability whether a detection is an object or not. This additional information reduces the effect of any adversarial perturbation. Therefore, we attack each class of Object Detection separately and train patches for each category, and use the latent contextual information used by the detection system to predict the class of the objects accurately.

Just as we did in classification, we tune various parameters, such as batch size,

learning rate. We found the perturbed patches for each class by back-propagating the gradient of sum of *class_loss* and *patch_loss* with respect the image, for all the images belonging to the specified class. We applied the perturbed patch to the image after masking and report the result of detection on the adversarial examples by running the Pascal evaluation scripts. To evaluate the performance of our patch on object detection, we compare the performance of mAP(mean average precision) over all classes and AP (average precision) of each class of the original images versus the adversarial images. We report the difference between the corresponding values for original and adversarial images.

5.2.1 Patch in center

We place the patch in the center of the image and evaluate the performance of YOLO on the adversarial input. For training we use batch size = 32 and learning rate = 1. The results of this experiment have been shown in table 5.3. The values in the table show the difference between the performance of original images on the pre-trained Yolo model and the performance of adversarial images on the same. Figure 5.4 vizualises this difference with a bar chart.

In the results table 5.3, the first row shows the change in precision in the attacked class. The highest change in Precision that we achieved is 21.32% for bird class, second highest for cat (15.1%) and third for horse (13.35%). The second row shows the average precision values of all classes achieved for the on applying the bird class patch, as this is the class having the best change in precision. The car class precision can be ignored in this case because the patch itself is detected as a car in several cases, which affects the precision of the car class.

Although we experimented with placing the patch randomly without obstructing any images, the results of the experiment were very poor.

method	aero	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	table
cls loss	11.29	3.55	**21.32**	7.48	6.03	5.86	0.61	15.1	2.88	9.51	6.05
best loss	24.48	2.84	21.32	23.83	5.75	1.93	43.81	15.56	4.95	10.53	5.61

method	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
cls loss	10.55	13.35	2.31	3.02	9.05	6.87	8.24	12.64	11.98	8.72
best loss	15.62	12.68	5.71	7.86	9.32	11.14	10.61	25.05	13.19	11.46

Table 5.3. Object detection results of our adversarial patch

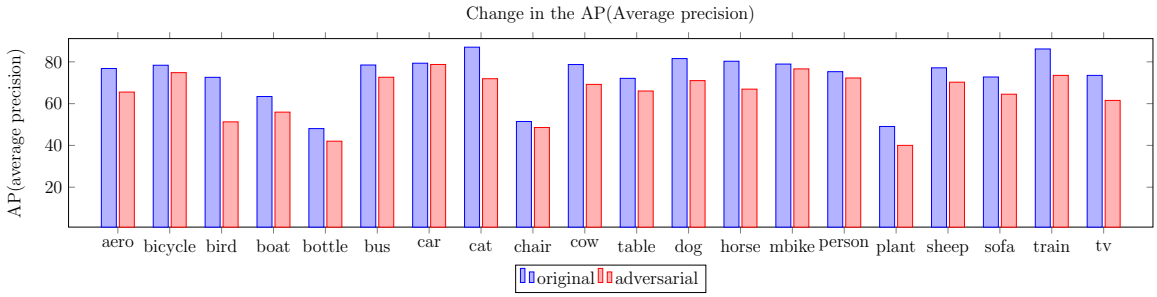


FIG. 5.4. Visualization of average precisions of the original Images versus adversarial images

The trained patches for each category are shown in Figure 5.5 and figure 5.6 shows the image results we achieve after applying the patch which works best (Patch obtained on attacking the bird class) on Pascal VOC dataset. 5.7 shows some negative examples for the same. A lot of the learned patches start look like other objects. This makes sense if we consider how the presence of an object changes the context of a scene. For example, in Figure 5.6 (6), the adversarial patch is above the aeroplanes, and the aeroplanes are not predicted after applying the patch, which looks like a car. In the real-world, you don't see a car flying above an aeroplane, and therefore by

virtue of placement of the patch, the prediction of the network is changed. Similarly, in Figure 5.6 (1), The patch itself is predicted as a car, but the train object is not predicted because the size of the car and the train do not fit the same context.

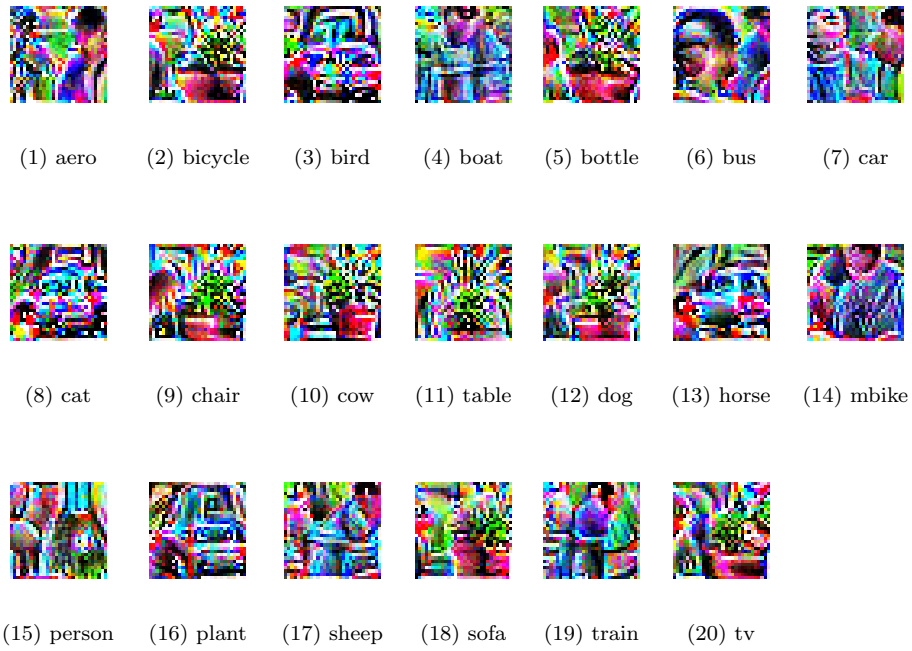
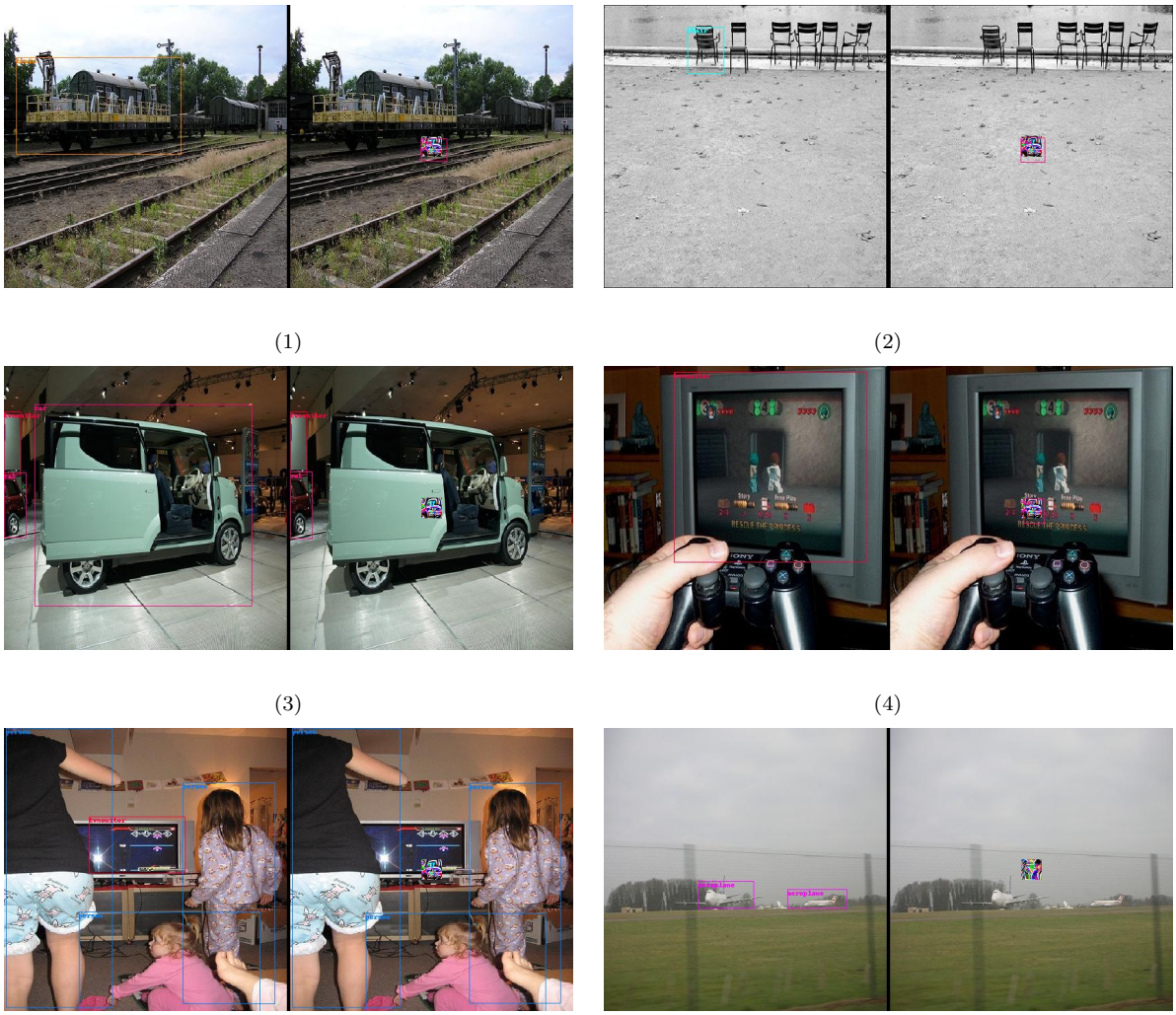


FIG. 5.5. Adversarial patches learned for each class in the Pascal-VOC dataset



(1) (2)

(3) (4)

(5) (6)

FIG. 5.6. Positive results of adversarial patches learned for object detection on the Pascal VOC dataset

(1)

(2)

FIG. 5.7. Negative results of adversarial patches learned for object detection on the Pascal VOC dataset

5.2.2 Importance of Context

It is interesting to see how attacking one class affects the other classes. We plotted the line graphs of average precision versus attacked class for some categories which we thought had similar context. By examining these graphs we can see that context is important for object detection. Figure 5.2.2 shows the change in average precision for the *aeroplane*, *bird* and *boat* categories for each class attack. We see that *aeroplane* and *bird* move in a very manner and whichever attack hurts *aeroplane* class

hurts bird class as well. We noticed that the *boat* class moves similar to *aeroplane* class too, and we think it might be due to the similarity in appearance and background color as the sky and the ocean are both blue.

The second graph, Figure 5.9 compares *dogs*, *cats* and *sheep* performance when attacked. As expected, *cat* and *dog* graphs lie close to each other and an attack on one affects the other. Figure 5.10 (1) compares the performance of *horse* and *cow* classes, which we thought may be comparable, but they are not as similar as we expected. This could be because humans occur on top of horses a lot of times but never on top of cows. After generating these three graphs, we noticed the similarity between *cow* and *sheep* class as well, as shown in Figure 5.10 (2).

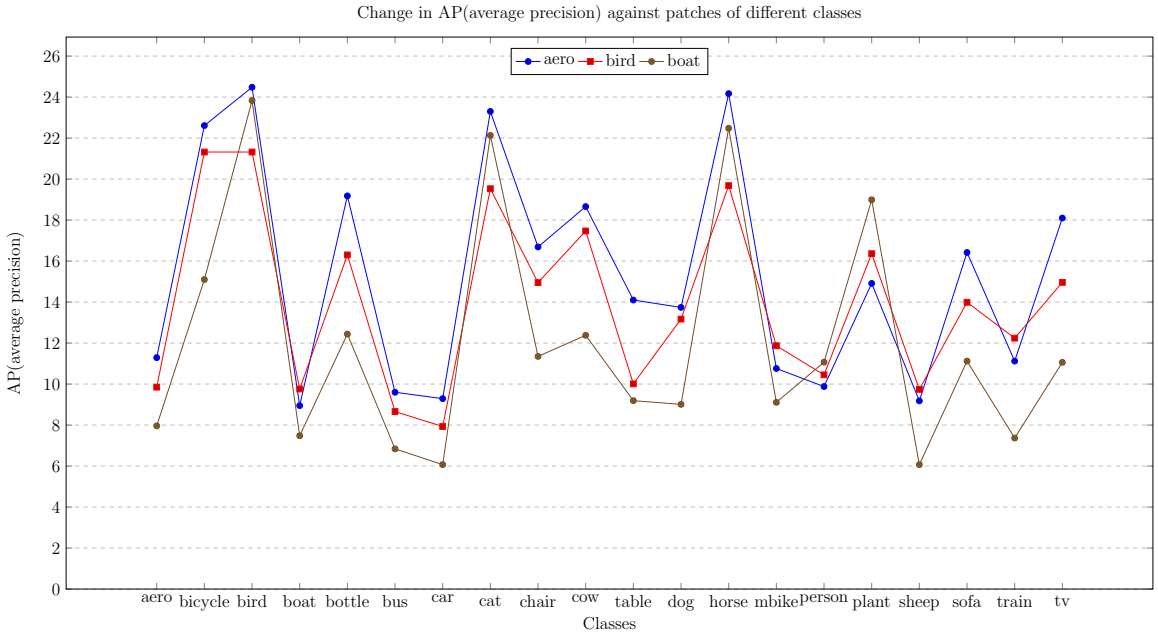


FIG. 5.8. Comparision of average precisions when attacking *aeroplane*, *bird* and *boat* classes

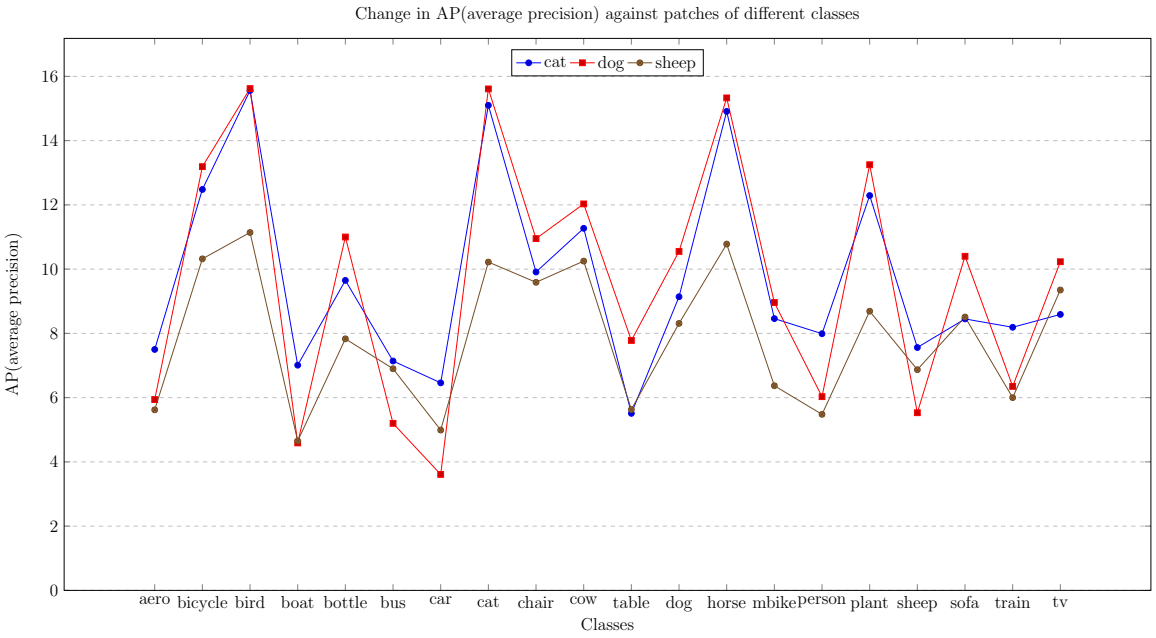
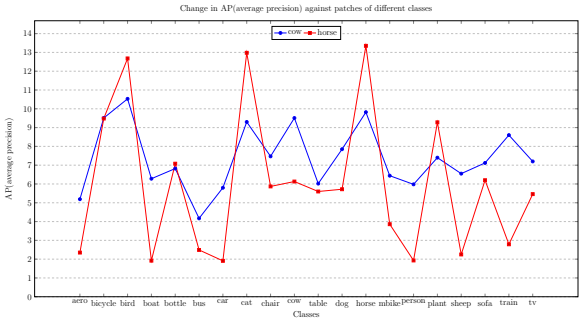
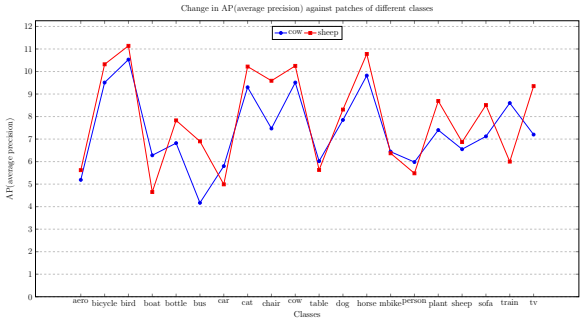


FIG. 5.9. Comparision of average precisions when attacking *cat*, *dog* and *sheep* classes



(1) *Cow* versus *Horse*



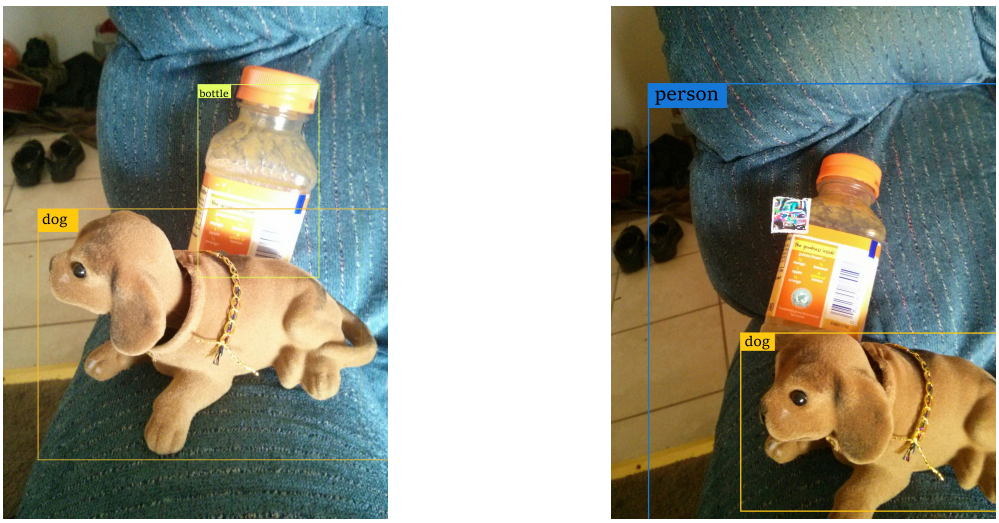
(2) *Cow* versus *Sheep*

FIG. 5.10. Comparison of average precisions of classes

5.3 Demonstration of adversarial examples in the physical world

So far we have shown how digitally generated adversarial images are able to fool the network. In this section we show that we can take a photo of an image containing an adversarial patch, and this photo is also able to confuse the network.

We obtained images from the classification and detection datasets which were able to fool the network. We then took photos of the original image and the photos of the adversarial images with a Nexus 4x phone, and passed these through the network. We found that more than 30% of the time, the adversarial photos images are also misclassified by the network. Figure **??** and 5.11 show examples of photographs taken of original images which are classified correctly, and photographs of adversarial examples which are misclassified.



(1) original image

(2) adversarial image)

FIG. 5.11. Real world adversarial example for fooling object detection

(1) original: stingray

(2) adversarial: purse

(3) original: toilet tissue, toilet paper, bathroom tissue

(4) adversarial: axolotl, mud puppy, Ambystoma mexicanum

FIG. 5.12. Real world adversarial examples for fooling classification

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, we presented a novel way of generating adversarial examples, by introducing a small patch in an image which can confuse image classification and object detection systems. For classification, placing the patch in the top-left corner of the image, we were able to change 42.1% of the network predictions from the testing data; while training and placing the patch in random locations along the edge of the image, we were able to fool the network 33% of the time. Although object detection we placed the patch in the center of the image, we were able to reduce the average precision of the bird class by 21.3%, cat by 15.1% and horse by 13.35%, which are our top three results for object detection. Along with this, we also demonstrated that cell-phone pictures of adversarial examples with the patch also fool the network.

6.2 Future Scope

There is a lot of scope for improvement and future work in this thesis. A way which could help improve the object detection performance could be to suppress the best detection for the targeted class by adding a max-pooling layer to the existing Yolo model. This loss function minimizing the probability of the best attacked object

detection combined with the exiting cross-entropy loss for classification could help increase the fooling ratio for object detection. Also, owing to the small size of the patch, this algorithm could be used to realize out real-time blackbox attacks against CNNs, as it will take lesser time to compute gradient with respect to 900 pixels (30x30) rather than 224x224 or 416x416 pixels.

It will be interesting to see performance of this algorithm on other classification and object detection models; performance of the same patch since it has been show n that adversarial examples are transferable, and also performance of the algorithm on other models. Lastly, using a patch may help to train the network better with respect to context and help the network to understand good and bad context.

REFERENCES

[1] Carlini, N., and Wagner, D. Defensive Distillation is Not Robust to Adversarial Examples.

[2] Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database.

[3] Everingham, M.; Ali Eslami, S. M.; Van Gool, L.; I Williams, C. K.; Winn, J.; Zisserman, A.; Everingham, M.; A Eslami, S. M.; Winn, J.; Van Gool Leuven, L. K.; Van Gool ETH, B. L.; K I Williams, S. C.; and Zisserman, A. 2015. The PASCAL Visual Object Classes Challenge: A Retrospective. *Int J Comput Vis* 111:98–136.

[4] Girshick, R.; Donahue, J.; Darrell, T.; and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation.

[5] Goodfellow, I. J.; Shlens, J.; and Szegedy, C. EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES.

[6] He, K.; Zhang, X.; Ren, S.; and Sun, J. Deep Residual Learning for Image Recognition.

[7] Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network.

[8] Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks.

[9] Kurakin, A.; Brain, G.; Goodfellow, I. J.; and Bengio, S. ADVERSARIAL MACHINE LEARNING AT SCALE.

[10] Kurakin, A.; Brain, G.; Goodfellow, I. J.; and Bengio, S. 2017. Workshop track -ICLR 2017 ADVERSARIAL EXAMPLES IN THE PHYSICAL WORLD.

[11] Lu, J.; Sibai, H.; Fabry, E.; and Forsyth, D. 2017. Standard detectors aren't (currently) fooled by physical adversarial stop signs.

[12] Lu, J.; Issaranon, T.; and Forsyth, D. SafetyNet: Detecting and Rejecting Adversarial Examples Robustly.

[13] Metzen, J. H.; Genewein, T.; Fischer, V.; and Bischoff, B. ON DETECTING ADVERSARIAL PERTURBATIONS.

[14] Moosavi-Dezfooli, S.-M.; Fawzi, A.; Fawzi, O.; and Frossard, P. 2016. Universal adversarial perturbations.

[15] Papernot, N.; Mcdaniel, P.; Wu, X.; Jha, S.; and Swami, A. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks.

[16] Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection.

[17] Simonyan, K., and Zisserman, A. 2015. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION.

[18] Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks.

[19] Tramèr, F.; Kurakin, A.; Papernot, N.; Boneh, D.; and Mcdaniel, P. Ensemble Adversarial Training: Attacks and Defenses.