On the Machine Illusion

by

Zhitao Gong

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
Feburary, 2019

Keywords: Adversarial, Security, Deep Learning, Computer Vision, Natural Language Processing

Approved by

Wei-Shinn Ku, Professor of Computer Science and Software Engineering
Xiao Qin, Professor of Computer Science and Software Engineering
Anh Nguyen, Assistant Professor of Computer Science and Software Engineering
Yang Zhou, Assistant Professor of Computer Science and Software Engineering
Shiwen Mao, Professor of Electrical and Computer Engineering

Abstract

In this work, we empirically study an emerging problem in machine learning community, i.e., the *adversarial samples*. Specifically, we focus on the discussion within the realm of neural networks. The existence of adversarial samples reveals yet another inconsistency in our hypothesis about neural networks. A adversarial sample is usually generated by adding very small and carefully chosen noise to a clean data sample, e.g., adding noise to an image to change some pixel values, replacing a few words in a sentence. Despite that they are almost the same (visually or semantically) as the clean samples from the perspective of human beings, the adversarial samples will trick a well-trained neural network into wrong predictions with very high confidence. In addition, we also show that adversarial samples exist in real world when the objects are in an unusual pose (e.g., a flipped-over school bus). We study this problem from two sides of the coin, i.e., defending against adversarial samples and generating adversarial samples. Concretely, to defend against adversarial samples, we propose a binary classification method to filter out adversarial samples. It achieves almost perfect accuracy on adversarial samples from seen distributions. However it fails to recognize adversarial samples from unseen distributions. To generate of adversarial samples, we first propose a framework to generate text adversarial samples for text classification problem (e.g., sentimental analysis). The framework generates high quality text adversarial samples. The limitation is that we do not have an explicit control over the semantics and syntax. In addition, we propose another framework to generate image adversarial samples by rendering 3D objects in unusual poses. It shows that natural adversarials in real world may exist in abundance. What's lacking in this dissertation is a theoretical exploration of this problem. We may revisit this problem when theories behind neural networks get matured.

Acknowledgments

I would like to thank Dr. Wei-Shinn Ku for his invaluable guidance during my research. Initially, when I barely had any experience for research, we had a meeting once a week where I learned how to think like a researcher and how to narrow down my areas of focus. After that, Dr. Ku gave me so much freedom and support in choosing my own areas and topics. I really appreciate Dr. Ku's patience and support during my graduate studies.

I would also like to thank Wenlu for her contribution in two of our work, i.e., classification methods against adversarial samples and generating text adversarials. She helped me with many useful experiments and discussions.

In addition, I would like to thank Dr. Anh Nguyen for inviting me to join his project, i.e., generating natural 3D adversarial images. I learned a lot from the team members, Michael, Qi and Chengfei. Especially I re-learned the idea of *moving fast and breaking things* from Michael.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Overview

Artificial intelligence (AI) helps us in many challenging tasks, including computer vision (e.g., image segmentation, object identification and recognition, image classification), natural language processing (e.g., machine translation (MT), question-and-answer (QA) system), recommendation systems, search, speech recognition, etc. The workhorse behind AI are the numerous machine learning models, including classical models (e.g., generalized linear models, SVM) and deep learning models, i.e., neural network-based models. Recently, deep learning models achieve state-of-the-art results in many fields.

However, [Sze+13] first shows that the state-of-the-art image models may be tricked into wrong predictions with high confidence when the input images are perturbed with carefully crafted noise. Furthermore, these perturbed images appear visually almost the same as the original ones from the perspective of human beings. These images are referred to as the *adversarial images*. Many followup work show that the adversarial samples are more universal than expected. Figure 1.1 demonstrates examples of the adversarial images on MNIST [LC10] dataset. As we can see, despite of the noise, we, as human beings, have no problem identifying the correct label for the digit. However, a well-trained neural network model (a simple convolution neural network (CNN) with test accuracy over 99% on clean test data) makes wrong predictions about the digit labels with high confidence.

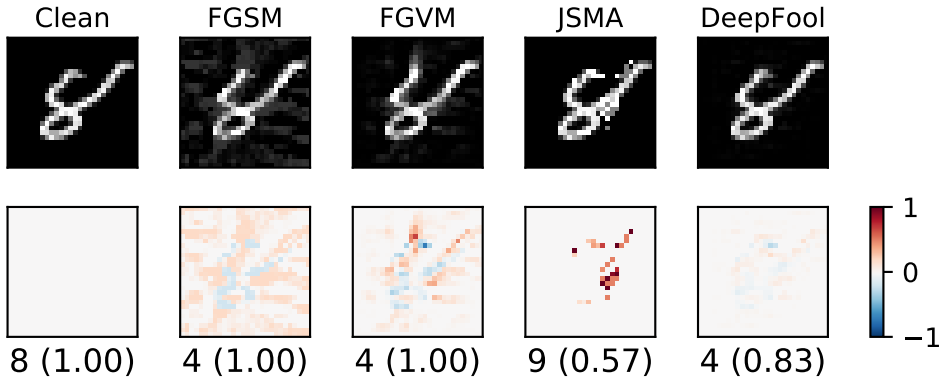


Figure 1.1: Adversarial images from a clean data sample in MNIST dataset [LC10]. The leftmost column is the clean image. The rest columns show 1) on the top, the adversarial images (top) and 2) on the bottom, the pixel difference between the adversarial image and the clean one. The labels on top of each column are the methods used to generate the adversarial samples. The labels below are the predictions by a well-trained neural network model (with test accuracy over 99% on clean test data) with probabilities (interpreted as confidence) in parenthesis. Note that the pixel values are normalized to $(0, 1)$ before being fed into the classification model. As a result, the noise value range is $(-1, 1)$.

1.2 Motivation

The investigation into this phenomenon has important applications both in practice and in theory. In the real word, adversarial examples pose a serious problem since more and more tasks are automated by neural networks models. Take the hateful comment filtering as an example. The replacement of a few seemingly non-important words in a sentence may turn a hateful comment into a "good" one, from the machine's point of view. This could potentially cause severe social problems. For some life-critical scenarios (e.g., autonomous driving, cancer diagnose), the lack of understanding in adversarial samples may put life at risk [Pre18]. As discussed in Section 5, neural network models may fail to recognize objects in weird poses. On the other hand, a deeper understanding of this adversarial phenomenon may advance our knowledge about neural networks. The theory of neural network is like a big jigsaw puzzle where the adversarial phenomenon is one important piece among many others, e.g., the generalization hypothesis, learning dynamics, non-convex optimization, properties

of the loss surface, etc. In this dissertation, we mainly focus on the empirical exploration of this problem.

1.3 Road Maps

In a series of work, we investigate both sides of the adversarial problem, i.e., defending against adversarial samples and generating adversarial samples.

1.3.1 Defending against Adversarial Samples

Observations from the image adversarial samples are that

1. The adversarial noise follows a specific direction [GSS14].
2. The neural nets are sensitive to individual pixel values [Sze+13].

Then it is natural to ask "can we utilize these properties to build a binary classifier to filter out adversarial samples?" The answer is *yes and no*. We propose a binary classification method to separate the adversarial samples from the clean ones. The results demonstrate that it works well for adversarial examples from seen distributions. However, there are also limitations to this binary classification approach where it fails to recognize the adversarial examples from unseen distributions. Please refer to Chapter 3 for a detailed discussion. This is based on our work [GWK17]. Note that another group proposed similar idea [Met+17] independently from us around the same time. We will have a brief discussion about the difference in our work.

1.3.2 Generating Adversarial Samples

First, we propose a framework to generate text adversarial samples. The difficulty of generating adversarial texts are two-folds.

1. The input space is discrete, which makes it difficult to perturb the input by accumulating small noise, as is common in generating adversarial images.

2. The quality evaluation of the generated texts are intrinsically difficult. Besides human evaluation, we have not yet found better ways to compare the quality of two text piece.

We propose a framework to workaround the first problem. Please refer to Chapter 4 for a detailed discussion. This is based on our work [Gon+18].

Second, we propose a framework to generate *natural* image adversarial samples. Instead of fiddling with the pixels in the input image, we render 3D objects in unusual poses to generate adversarial images. The implication of this work is that adversarial examples may exist in abundance in the natural world. This is based on our work [Alc+18]. Note that despite that I include the whole work in this thesis for completeness, I contributed to only part of the experiments and discussion. Concretely, Michael contributed to the non-differentiable renderer experiments and analysis, Qi prepared the dataset, Chengfei contributed to some analysis, and I contributed to the differentiable renderer experiments and analysis. Michael and Qi developed the desktop version of the software release with the paper, while I prepared the web version. Please refer to Chapter 5 for a detailed discussion.

Chapter 2

Background

We briefly introduce some background knowledge about neural networks in Section 2.1, please refer to [Goo+16] for a more thorough discussion. In addition, we will discuss some adversarial algorithms in image domain in Section 2.2. These algorithms demonstrates different properties of adversarial examples.

2.1 Neural Networks

Neural network is a connectionism model comprising of artificial neurons that are interconnected in a certain pattern. The neuron was originally inspired by the biological neurons, as shown in Figure 2.1. Despite of the widely used analogy between biological and artificial neurons, they are, in effect, very different from each other in the way they work. In the following context, we use neurons to refer to artificial neurons.

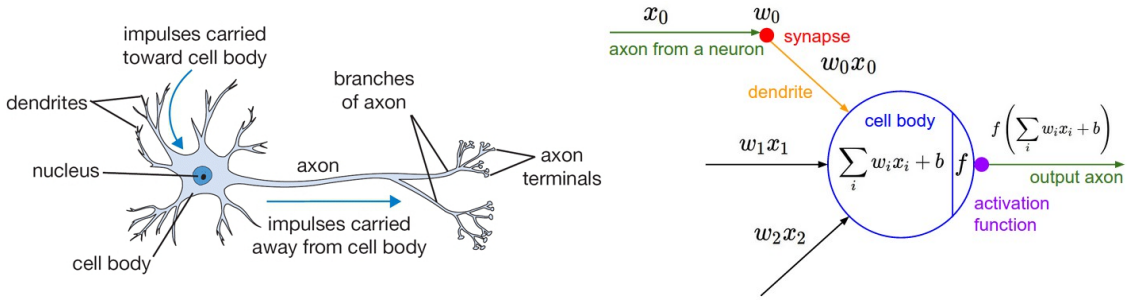


Figure 2.1: The artificial neuron (right) is inspired by the biological neurons (left). For the artificial neuron on the right, The x's are signals from other neurons, weights w's controls the signal strength. The cell is essentially a function f that aggregates and transforms the inputs to produce the signal. (image credit: cs231n)

Simply put, A neuron is just a function. It can be simple math functions, e.g., identity function $f(x) = x$, sigmoid function $f(x) = 1/(1 + e^{-x})$, rectified linear units (ReLU)

$f(x) = \max(0, x)$, etc. Some neurons may be more complex. Recurrent cells have feedback loops, i.e., output being fed into the input (the dotted line in Figure 2.2). Some neurons may have internal parameters, e.g., filters in the convolution network ($f(X) = sum(M \bigotimes X)$ where $\bigotimes$ denotes element-wise multiplication, M is a weight matrix of the same size as X). More generally, each neuron may also be a small neural network [LCY13]. Depending on the level of granularity, neurons may take different forms. The simple ones may be used as motifs to construct more complex ones [ZL16].

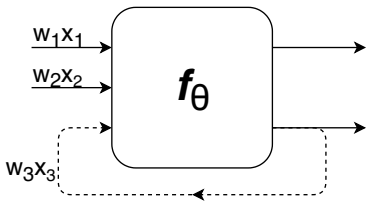


Figure 2.2: A general illustration of a neuron, which is a function. Some neurons may have internal parameters, e.g., filters in the convolution network. Others may have feedback loop, i.e., dotted lines in the figure, e.g., LSTM cells.

Connecting these neurons in a certain pattern, it forms a neural network system. In a practical neural network, there are usually millions of weights. The training of neural network is the process of obtaining a *suitable* value for each of the weight. Despite that the training procedure is usually formulated as a non-convex optimization process where the target is to minimize some loss term L, the gradient descent (GD) algorithms work pretty good in practice, e.g., first-order GD (e.g., stochastic gradient descent (SGD), Adam [KB14]), second-order GD (e.g., Levenberg–Marquardt (LM) [Mor78]). The training algorithm is usually referred to as the error back-propagation [RHW86]. It is, in effect, an application of the chain rule. The error backpropagation for a single neuron is illustrated in Figure 2.3.

2.2 Adversarial Images

In this chapter, we briefly discuss some of the adversarial generating algorithms in the image domain. Each algorithm represents a different view of the adversarial samples.

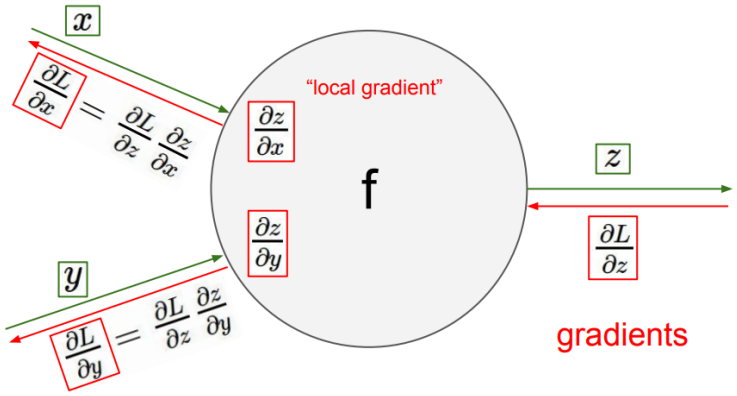


Figure 2.3: The error backpropagation in a single neuron. The green lines are forward pass, while the red lines are backward pass, i.e., the back-propagation. The error L is passed back through the system following the chain rule to calculate each weight's contribution to the error term, e.g., $\frac{\partial L}{\partial x}$, $\frac{\partial L}{\partial y}$. (Image credit cs231n)

Besides, these algorithms demonstrate the basic ideas on how to find adversarial samples (or noises) for a target model. Adversarial algorithms in other domains (e.g., sentiment analysis, speech recognition) are generally variants of these algorithms, and will be surveyed in the following chapters where necessary.

2.2.1 Notations

In this section, we will introduction some general notations used in this dissertation. Notations specific to one algorithm or certain part of the analysis are introduced in each section respectively.

We denote a well-trained classifier by $f : R^m \to \{1 \dots k\}$ which maps an m-dimensional input vector to a discrete label set. We denote the input by $x \in R^m$, output $\overline{y} \in [0, 1]^k$, i.e., $\overline{y} = f(x)$. The ground truth for x is denoted by y. Note that we usually have an probabilistic interpretation for the output y. if we have $k = 2$, i.e., binary classification, then the output layer is usually a sigmoid function, and y represents the probability of x belonging to class 1. If $k > 2$, i.e., multi-label classification, the output layer is usually passed through a softmax function which generates a probability distribution over the labels, i.e., $\sum_{\mathrm{i}} \mathrm{y_i} = 1$. And y_i (the i-th component of y) represents the probability of x belonging to i-th category.

If we denote the predicted label for x by l_x, then

1. In the binary classification case $l = 1$ if $y > 0.5$ and 0 otherwise.

2. In the multi-label classification case, $l_x = \arg\max_i\{y_i\}$. In this case, the label is usually represented by one-hot encoding. For example if $k = 3$, and the true label for x is 1, then $y = [0, 1, 0]$.

Abusing the notation a little bit wherever the meaning is clear, we use $f(x) \neq f(\widetilde{x})$ to denote that the model f predicts different labels for x and $\widetilde{x}$.

In addition, we assume that the classifier f is associated with a continuous loss function $J(y, \overline{y})$. Common choices for the loss function J are cross-entropy loss, mean squared error loss (MSE), etc.

Although, for simplicity, we illustrate the adversarial algorithms with a vector as the input, the algorithms and their variants can be applied to tensor input as well, e.g., an image input of size $W \times H \times C$.

We use δx to denote the small perturbation found by the adversarial algorithms. The adversarial sample for x is denoted by $\widetilde{x} = x + \delta x$. Some algorithms are targeted attack, i.e., allowing to change the prediction of $\widetilde{x}$ to a user-defined label, which is denoted as $\widetilde{y}$.

2.2.2 Problem Formulation

Generally speaking, we want to find a *small* perturbation δx for an input x such that a well-trained classifier f, that correctly predicts x, will produce a wrong prediction for $\widetilde{x}$, i.e., $f(x) \neq f(\widetilde{x})$ (e.g., see 1.1).

The above is a very broad definition, different algorithms may formulate the problem in different ways. Besides, regarding the definition of δx being *small*, there is no hard criterion, nor a widely accepted measurement. In literature, the L_1-, L_2-, L_∞-norms are usually used as the measurement when generating and comparing different adversarial algorithms. The

general guideline is that the perturbed inputs $\widetilde{x}$ should not interfere the judgment of human beings.

2.2.3 Overview of Generating Adversarial Images

Generally speaking, the proposed adversarial generating methods in literature fall into two strategies based on its intuition, where the core idea of the first category is to move data points around till its label changes, and the other one is to create a mapping between clean and adversarial samples (or noises).

Move Data Points

Essentially, this class of methods move the data points a small step along a carefully chosen direction. It has been shown that it is very unlikely to arrive at adversarial samples simple by a random walk [Sze+13].

1. The direction may be where the loss of the clean samples increases, e.g., FGSM [GSS14] and its variants, FGSM (without label leaking) [KGB16a], iterative FGSM [KGB16b], FGVM [Miy+15]. It may also be the direction where the loss of adversarial samples decreases, e.g., constrained optimization based [Sze+13].

2. The direction may also be where the probability of the correct label decreases (or the probabilities of the target label increases), e.g., JSMA [Pap+15b], CW [CW16].

3. It could also be the direction towards the decision boundary e.g., DeepFool [MFF15], one-pixel attack [SVK17], so that one tiny nudge across the boundary would create an adversarial sample of a different category.

Map Clean Samples to Adversarial

This class of methods are relatively less explored. Adversarial transformation network (ATN) [BF17] employs an autoencoder to generate adversarial samples or noises. [Xia+18;

ZDS17] employs a generative model (i.e., GAN [Goo+14]) to map from clean samples to adversarial ones. The advantages of this class of methods are (i) that the generation is usually fast since only one pass of forward computation is needed, and (ii) that the adversarial sames may be of great diversity if a generative network is used.

2.2.4 Case Studies

Constrained Optimization Approach

[Sze+13] first explored the adversarial images following a constrained optimization formulation. Concretely, the authors aim to

- Minimize $\|\delta x\|_2$ subject to

  1. $f(\widetilde{x}) = l$
  2. $\widetilde{x} \in [0,1]^m$

Note that the task in this formulation is non-trivial only if $l_{\widetilde{x}} \neq l_x$. The second constraint guarantees that the $\widetilde{x}$ is a valid image. There might exist many valid minimizers and the exact solution may be computationally prohibitive. The authors proposed an approximation.

- Minimize $c\|\delta x\| + J(f(\widetilde{x}), \widetilde{y})$ subject to $\widetilde{x} \in [0,1]^m$.

Where the scale c is determined by line search. The author use L-BFGS to solve This constrained optimization. The downside of this formulation is that

1. there are not many choices for the optimization methods since most only work in unconstrained case, and
2. the constrained optimization take longer.

The Carlini-Wagner (CW) [CW16] uses a reparameterization trick to turn it into an unconstrained optimization. See Section 2.2.4.

Fast Gradient Method (FGM)

Fast gradient method is a class of method that relies on the $\nabla_x J$.

1. Fast Gradient Sign Method (FGSM) [GSS14] proposes the original fast gradient sign method (FGSM) based on the hypothesis that *neural networks are too linear to resist linear adversarial perturbation.* Concretely, we have

$$\widetilde{x} = x + \epsilon \operatorname{sign}\{\nabla_x J(x, y)\} \tag{2.1}$$

   The intuition is that FGSM tries to modify the input towards the direction where the classification loss for this data sample increases. It is referred to as the *fast method* in [KGB16a].

   However, in Equation 2.1, the adversarial sample is generated with its true label y, which is assumed not known in practice. This is referred to as the label leaking problem [KGB16b]. This work [KGB16b] introduces a revised version of FGSM, where the true label y is replaced by the *predicted* label (i.e., $\overline{y}$) when generating adversarial examples. The revised FGSM is as follows.

$$\widetilde{x} = x + \epsilon \operatorname{sign}\{\nabla_x J(x, \overline{y})\} \tag{2.2}$$

2. Fast Gradient Value Method (FGVM) A simple variant of FGSM is fast gradient *value* method (FGVM) [Miy+15], where the gradient values are directly used when computing the noise, instead of the sign of gradients.

$$\widetilde{x} = x + \epsilon \nabla_x J(x, \overline{y}) \tag{2.3}$$

   In practice, the FGSM works better, i.e., generating more subtle noise, than FGVM.

3. Iterative Fast Gradient Method An intuitive way to extent FGSM is the *iterative* FGSM [KGB16a], where the authors apply it multiple times with small step size, and clip pixel values of intermediate results after each step to ensure that they are in an ϵ-neighborhood of the original image. Concretely, we have

$$\widetilde{x}_0 = x, \qquad \widetilde{x}_{n+1} = \mathrm{CLIP}_{x,\epsilon}\{\widetilde{x}_n + \alpha\,\mathrm{sign}\{\nabla_{\widetilde{x}_n} J(\widetilde{x}_n, f(\widetilde{x}_n))\}\} \tag{2.4}$$

4. Least-likely Class Method The above variants of FGSM so far only increases the cost of correct class, without specifying a desired target class. For classification task with finer labels (e.g., different breeds of dogs in ImageNet), the above method may create uninteresting adversarial samples. In order to create more interesting misclassification, [KGB16a] proposes to modify the image towards the direction where the probability for the least-likely class is increased. The least-likely is defined by

$$y_{ll} = \arg\min_i p(y_i \mid x) \tag{2.5}$$

The intuition is that for a well-trained classifier, the least-likely class should be high different from the true class. Concretely we have

$$\widetilde{x}_0 = x, \qquad \widetilde{x}_{n+1} = \mathrm{CLIP}_{x,\epsilon}\{\widetilde{x}_n - \alpha\,\mathrm{sign}\{\nabla_{\widetilde{x}_n} J(\widetilde{x}_n, y_{ll})\}\} \tag{2.6}$$

Notice the minus sign in the above equation. We want to increase the probability instead of decreasing it. An easy extension to the least-likely class method is to use desired target class instead of the least-likely one. This is a more generalized version of this method.

Jacobian-based Saliency Map Approach (JSMA)

Similar to the target class method, JSMA [Pap+15b] allows to specify the desired target class. However, instead of adding noise to the whole input, JSMA changes only one pixel at a time. A *saliency score* is calculated for each pixel where the pixel with the highest score is chosen to be perturbed.

$$
\begin{aligned}
s(i) &= \begin{cases} 0 & \text{if } s_t < 0 \text{ or } s_o > 0 \\ s_t^{(i)} \left| s_o^{(i)} \right| & \text{otherwise} \end{cases} \\
s_t^{(i)} &= \frac{\partial \widetilde{y}_t}{\partial x_i} \qquad s_o^{(i)} = \sum_{k \neq t} \frac{\partial \widetilde{y}_k}{\partial x_i}
\end{aligned}
\tag{2.7}
$$

Where t denotes the target class, s_t is the Jacobian value of the desired target class y_t with regard to each individual pixel, s_o is the sum of Jacobian values of all non-target classes. The authors hypothesis is that the saliency score indicates the sensitivity of each output class with regard to each individual pixel, or how much the probability for each class will change when we perturb a pixel. With this information, we want to perturb the pixel towards the direction where $p(\widetilde{y}_t \mid x)$ increases the most.

The pixel value is either increase to maximum (i.e., 1.0) or decreased to minimum (i.e., 0.0).

In the paper, the authors calculate scores for pairs of pixels, instead of individual pixel. The saliency score for a pixel pair is defined as

$$
s(i,j) = (s_t^{(i)} + s_t^{(j)}) \times \left| s_o^{(i)} + s_o^{(j)} \right| \tag{2.8}
$$

The reason was that pixel pairs give better performance in practice. However, note that the computation for pixel pairs is $O(n^2)$ since we need to compute scores for every pixel pair. In practice, JSMA may be slow for large images.

There is also a small implementation detail. In the original paper, the authors used logits, i.e., the values before the softmax layer to calculate the gradients. However, in the distillation defense [Pap+15a], the authors used the output of softmax layer to calculate the gradients. Despite this discrepancy, the two versions perform similarly.

Carlini-Wagner (CW)

Carlini-Wagner [CW16] method is a more friendly version of the constrained optimization method discussed in Section 2.2.4. The problem was formulated as such.

$$\begin{aligned} \text{minimize} \quad & D(x, \widetilde{x}) \\ \text{subject to} \quad & f(\widetilde{x}) = \widetilde{y},\ \widetilde{x} \in [0,1]^m \end{aligned} \tag{2.9}$$

Where D is some distance metric, e.g., L_0-, L_2-, L_∞-norm. As discussed before, the problem with this formulation are the constraints. The authors propose to optimize a delegate object function such that $f(\widetilde{x}) = t$ if and only if $g(\widetilde{x}) \leq 0$. With this delegated object function, the original problem is formulated as such.

$$\begin{aligned} \text{minimize} \quad & D(x, \widetilde{x}) \\ \text{subject to} \quad & g(\widetilde{x}) \leq 0,\ \widetilde{x} \in [0,1]^m \end{aligned} \tag{2.10}$$

Which can be alternatively formulated as

$$\begin{aligned} \text{minimize} \quad & D(x, \widetilde{x}) + cg(\widetilde{x}) \\ \text{subject to} \quad & \widetilde{x} \in [0,1]^m \end{aligned} \tag{2.11}$$

Where $c > 0$ is a *suitably chosen constant*.

DeepFool

DeepFool [MFF15] moves the data point along the direction to the nearest decision boundary. The intuition is that if you move the data point just cross the decision boundary, you would be in principle create an adversarial with the minimum distortion. Concretely we have

$$\widetilde{x} = x + \epsilon r \tag{2.12}$$

Where r is the approximated vector to the nearest decision boundary, ϵ is chosen to take a small value, e.g., 1.04. In other words, $x + r$ lies on the decision boundary, and $x + \epsilon r$ is the point just across the decision boundary. The r is calculated iteratively.

DEEPFOOL(f, x)

$x_0 = x$
$i = 0$
while $l_{x_i} = l_x$
    for $k \neq l_x$
        $a_k = \nabla f_k(x_i) - \nabla f_{l_x}(x_i)$
        $b_k = f_k(x_i) - f_{l_x}(x_i)$
    $t = \arg\min_{k \neq l_x} \frac{|b_k|}{\|a_k\|_2}$
    $r_i = \frac{|b_t|}{\|a_t\|_2} a_t$
    $x_{i+1} = x_i + r_i$
    $i = i + 1$
return $r = \sum r_i$

In 2D dimension, $\frac{a_t}{\|a_t\|_2}$ is the unit direction to the nearest decision boundary, and $|b_t|$ is the distance to the nearest decision boundary. This, however, does not apply to higher dimensions. The authors propose to workaround this problem by repeating this procedure until reaching the decision hyperplane. It is rather difficult to reason that we will get to the

decision hyperplane in the optimal direction. In practice, however, we do get very subtle noise.

Adversarial Transformation Network (ATN)

A complete different idea is proposed in [BF17], i.e., the adversarial transformation network (ATN). Instead of a routine-based approach where we repeat the procedure for each coming sample, the authors uses build a model mapping from the normal image to its corresponding adversarial sample. Suppose we have a model $g : x \to \widetilde{x}$, the model is trained by solving the following optimizations

$$L_g = \sum_{x_i \in \mathcal{X}} \beta L_{\mathcal{X}}(g(x_i), x_i) + L_{\mathcal{Y}}(f(g(x_i)), f(x_i)) \tag{2.13}$$

Where the $L_{\mathcal{X}}$ is a loss term in the input space, it encourages the adversarial image to be *close* to the original one, e.g., L_2 loss, perceptual similarity loss [JAF16], etc. And $L_{\mathcal{Y}}$ is a loss term in the output space of f that encourages misclassification of the $\widetilde{x}$. Concretely, $L_{\mathcal{Y}}$ takes the form of

$$\begin{aligned} L_{\mathcal{Y}} &= L_2(f(\widetilde{x}), r(f(x), t)) \\ r(y, t) &= \text{norm}\left(\begin{cases} \alpha \max y & \text{if } k = t \\ y_k & \text{otherwise} \end{cases} \right) \end{aligned} \tag{2.14}$$

Where α is a hyper-parameter, r is referred to as the re-ranking function, which increases the probability of the target label t to $\alpha \max y$ while keeping the other unchanged. The *norm* function normalizes the target $\widetilde{y}$ to make sure that it is a valid probability distribution. Intuitively speaking, $L_{\mathcal{Y}}$ increases the target label probability, while keeping the order and relative scale of all the probabilities for other labels.

Chapter 3

Defend against Image Adversarial Samples

3.1 Introduction

As 1.1 shows, although adversarial and clean images appear visually indiscernible, their subtle differences can successfully fool the deep neural networks. The observations are Observations from the image adversarial samples are that

1. The adversarial noise follows a specific direction [GSS14].

2. The neural nets are sensitive to individual pixel values [Sze+13].

s a result, an intuitive question is: whether it is possible to leverage the network's sensitivity to subtle differences to distinguish between adversarial and clean images? In this work, we explore this intuition and demonstrate that a simple binary classifier can separate the adversarial from the original clean images with very high accuracy (over 99%). So the answer to the above question is *yes*. However, we also notice that the binary classifier approach suffers from the *generalization issue*.

1. it is sensitive to hyper-parameters used in crafting adversarial images, e.g., ϵ in fast gradient method, and

2. it is sensitive to different adversarial crafting algorithms.

In other words, different algorithms will generate adversarial images that follow different distributions. It is insufficient to train the classifier only on one type of adversarial samples. In addition to that, we also showed that this limitation is shared among other proposed

defense methods against adversarial images, e.g., adversarial training [Hua+15; KGB16a], defensive knowledge distillation (KD) [Pap+15a], etc.

Our key contributions are:

1. We show that binary classifier can successfully separate adversarial from clean samples that follow similar distributions.

2. In addition, the binary classifier are robust to second-round adversarial attack, in other words, it is difficult to bypass the classifier with adversarial samples that fools the protected model.

3. However, we also show that currently proposed defense methods, including our binary classifier approach, does not generalize to adversarial samples that follow different distributions, e.g., created from different methods.

This chapter is organized as follows. In Section 3.2, we give an overview of the current literature in defending against adversarial images (the generating algorithms are surveyed in Section 2.2). The procedure for our investigation is outlined in Section 3.3. Section 3.4 presents our experiment results and detailed discussions. And we conclude this chapter in Section 3.5.

3.2 Related Work

The adversarial images have been shown to be transferable among deep neural networks [Sze+13; KGB16a]. This poses a great threat to current learning systems in that the attacker needs not the knowledge of the target system. Instead, the attacker can train a different model to create adversarial samples which are still effective for the target deep neural networks. What's worse, [PMG16] has shown that adversarial samples are even transferable among different machine learning techniques, e.g., deep neural networks, support vector machine, decision tree, logistic regression, etc.

Small steps have been made towards the defense of adversarial images. [KGB16b] shows that some image transformations, e.g., Gaussian noise, Gaussian filter, JPEG compression, etc., can effectively recover over 80% of the adversarial images. However, in our experiment, the image transformation defense does not perform well on images with low resolution, e.g., MNIST. Knowledge distillation (KD) [HVD15] is also shown to be an effective method against most adversarial images [Pap+15a]. The restrictions of defensive knowledge distillation are (i) that it only applies to models that produce categorical probabilities, (ii) that it needs model re-training. Adversarial training [KGB16b; Hua+15] was also shown to greatly enhance the model robustness to adversarials. However, as discussed in Section 3.4.2, defensive distillation and adversarial training suffers from, what we call, the generalization limitations. Our experiment suggests this seems to be an intrinsic property of adversarial datasets.

3.3 Method

Generally, we follow the steps below to evaluate the effectiveness and limitations of the binary classifier approach.

1. Train a deep neural network f_1 on the original clean training data X_{train}, and craft adversarial dataset from the original clean data, $X_{train} \rightarrow X_{train}^{adv(f_1)}$, $X_{test} \rightarrow X_{test}^{adv(f_1)}$, where $X^{adv(f_1)}$ denotes adversarial examples created from X targeting model f_1. Here, f_1 is the target model that we want to protect.

2. Train a binary classifier f_2 on the combined (and shuffled) training data $\{X_{train}, X_{train}^{adv(f_1)}\}$, where X_{train} is labeled 0 and $X_{train}^{adv(f_1)}$ labeled 1.

3. Test the accuracy of f_2 on X_{test} and $X_{test}^{adv(f_1)}$, respectively. This will show the effectiveness of the binary classifier approach.

4. Construct second-round adversarial test data, $\{X_{test}, X_{test}^{adv(f_1)}\} \rightarrow \{X_{test}, X_{test}^{adv(f_1)}\}^{adv(f_2)}$ and test f_2 accuracy on this new adversarial dataset. Intuitively, we want to investigate whether we could find adversarial samples (i) that can successfully bypass the binary classifier, and (ii) that can still fool the target model if they bypass the binary classifier. Since adversarial datasets are shown to be transferable among different machine learning techniques [PMG16], the binary classifier approach will be seriously flawed if f_2 failed the above second-round attacking test.

3.4 Experiment

The code to reproduce our experiment are available at `https://github.com/gongzhitaao/adversarial-classifier`.

3.4.1 Efficiency and Robustness of the Classifier

	f_1		f_2			
Dataset	X_{test}	$X_{test}^{adv(f_1)}$	X_{test}	$X_{test}^{adv(f_1)}$	$\{X_{test}\}^{adv(f_2)}$	$\{X_{test}^{adv(f_1)}\}^{adv(f_2)}$
MNIST	0.9914	0.0213	1.00	1.00	0.00	1.00
CIFAR10	0.8279	0.1500	0.99	1.00	0.01	1.00
SVHN	0.9378	0.2453	1.00	1.00	0.00	1.00

Table 3.1: Accuracy on adversarial samples generated with FGSM/TGSM.

We evaluate the binary classifier approach on MNIST, CIFAR10, and SVHN datasets. Of all the datasets, the binary classifier achieved accuracy over 99% and was shown to be robust to a second-round adversarial attack. The results are summarized in Table 3.1. Each column denotes the model accuracy on the corresponding dataset. The direct conclusions from Table 3.1 are summarized as follows.

1. Accuracy on X_{test} and $X_{test}^{adv(f_1)}$ suggests that the binary classifier is very effective at separating adversarial from clean dataset. Actually during our experiment, the accuracy

on X_{test} is always near 1, while the accuracy on $X_{test}^{adv(f_1)}$ is either near 1 (successful) or near 0 (unsuccessful). Which means that the classifier either successfully detects the subtle difference completely or fails completely. We did not observe any values in between.

2. Accuracy on $\{X_{test}^{adv(f_1)}\}^{adv(f_2)}$ suggests that we were not successful in disguising adversarial samples to bypass the the classifier. In other words, the binary classifier approach is robust to a second-round adversarial attack.

3. Accuracy on $\{X_{test}\}^{adv(f_2)}$ suggests that in case of the second-round attack, the binary classifier has very high false negative. In other words, it tends to recognize them all as adversarials. This, does not pose a problem in our opinion. Since our main focus is to block adversarial samples.

3.4.2 Generalization Limitation

Before we conclude too optimistic about the binary classifier approach performance, however, we discover that it suffers from the *generalization limitation*.

1. When trained to recognize adversarial dataset generated via FGSM/TGSM, the binary classifier is sensitive to the hyper-parameter ϵ.

2. The binary classifier is also sensitive to the adversarial crafting algorithm.

In out experiment, the aforementioned limitations also apply to adversarial training [KGB16b; Hua+15] and defensive distillation [Pap+15a].

Sensitivity to ϵ

Table 3.2 summarizes our tests on CIFAR10. For brevity, we use $f_2\big|_{\epsilon=\epsilon_0}$ to denote that the classifier f_2 is trained on adversarial data generated on f_1 with $\epsilon = \epsilon_0$. The binary classifier is trained on mixed clean data and adversarial dataset which is generated via FGSM

with $\epsilon = 0.03$. Then we re-generate adversarial dataset via FGSM/TGSM with different ϵ values.

	$f_2\big\|_{\epsilon=0.03}$	
ϵ	X_{test}	$X_{test}^{adv(f_1)}$
0.3	0.9996	1.0000
0.1	0.9996	1.0000
0.03	0.9996	0.9997
0.01	0.9996	**0.0030**

Table 3.2: ϵ sensitivity on CIFAR10

As shown in Table 3.2, $f_2\big|_{\epsilon=\epsilon_0}$ can correctly filter out adversarial dataset generated with $\epsilon \geq \epsilon_0$, but fails when adversarial data are generated with $\epsilon < \epsilon_1$. Results on MNIST and SVHN are similar. This phenomenon was also observed in defensive training [KGB16b]. To overcome this issue, they proposed to use mixed ϵ values to generate the adversarial datasets. However, Table 3.2 suggests that adversarial datasets generated with smaller ϵ are *superset* of those generated with larger ϵ. This hypothesis could be well explained by the linearity hypothesis [KGB16a; WG16]. The same conclusion also applies to adversarial training. In our experiment, the results of defensive training are similar to the binary classifier approach.

Disparity among Adversarial Samples

In our experiment, we also discovered that the binary classifier is sensitive to the algorithms used to generate the adversarial datasets.

Specifically, the binary classifier, that is trained on FGSM adversarial dataset achieves good accuracy (over 99%) on FGSM adversarial dataset, but not on adversarial generated via JSMA, and vise versa. However, when binary classifier is trained on a mixed adversarial samples dataset from FGSM and JSMA, it performs well (with accuracy over 99%) on both datasets. This suggests that FGSM and JSMA generate adversarial datasets whose distributions are *far away* from each other. It is too vague without defining precisely what

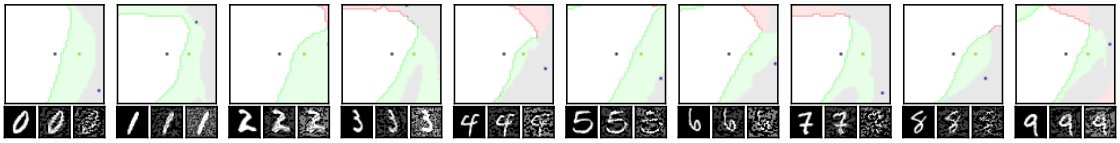


Figure 3.1: Adversarial training [Hua+15; KGB16b] does not work. This is a church window plot [WG16]. Each pixel (i, j) (row index and column index pair) represents a data point $\tilde{x}$ in the input space and $\tilde{x} = x + \mathbf{h}\epsilon_j + \mathbf{v}\epsilon_i$, where $\mathbf{h}$ is the direction computed by FGSM and $\mathbf{v}$ is a random direction orthogonal to $\mathbf{h}$. The ϵ ranges from $[-0.5, 0.5]$ and $\epsilon_{(\cdot)}$ is the interpolated value in between. The central black dot • represents the original data point x, the orange dot (on the right of the center dot) • represents the last adversarial sample created from x via FGSM that is used in the adversarial training and the blue dot • represents a random adversarial sample created from x that cannot be recognized with adversarial training. The three digits below each image, from left to right, are the data samples that correspond to the black dot, orange dot and blue dot, respectively. □ ( ) represents the data samples that are always correctly (incorrectly) recognized by the model. represents the adversarial samples that can be correctly recognized without adversarial training only. And represents the data points that were correctly recognized with adversarial training only, i.e., the side effect of adversarial training. (Image credit: [GWK17])

is *being far away*. In our opinion, they are *far away* in the same way that CIFAR10 is *far away* from SVHN. A well-trained model on CIFAR10 will perform poorly on SVHN, and vise versa. However, a well-trained model on the the mixed dataset of CIFAR10 and SVHN will perform just as well, if not better, on both datasets, as if it is trained solely on one dataset.

The adversarial datasets generated via FGSM and TGSM are, however, *compatible* with each other. In other words, the classifier trained on one adversarial datasets performs well on adversarials from the other algorithm. They are compatible in the same way that training set and test set are compatible. Usually we expect a model, when properly trained, should generalize well to the unseen data that follow the same distribution, e.g., the test dataset.

In effect, it is not just FGSM and JSMA are incompatible. We can generate adversarial data samples by a linear combination of the direction computed by FGSM and another random orthogonal direction, as illustrated in a church plot [WG16] (see Figure 3.1). Figure 3.1 visually shows the effect of adversarial training [KGB16b]. The pixels in each image represents adversarial samples generated from *one* data sample, which is represented as a black dot in the center of each image. The last adversarial sample used in adversarial training is

represented as an orange dot (on the right of black dot, i.e., in the direction computed by FGSM). The green area represents the adversarial samples that cannot be correctly recognized without adversarial training but can be correctly recognized with adversarial training. The red area represents data samples that can be correctly recognized without adversarial training but cannot be correctly recognized with adversarial training. In other words, it represents the side effect of the adversarial training, i.e., slightly reducing the model accuracy. The white (gray) area represents the data samples that are always correctly (incorrectly) recognized with or without adversarial training.

As we can see from Figure 3.1, adversarial training does make the model more robust against the adversarial samples (and adversarial samples around it to some extent) used for training (green area). However, it does not rule out all adversarials. There are still adversarial samples (gray area) that are not affected by the adversarial training. Further more, we could observe that the green area largely distributes along the horizontal direction, i.e., the FGSM direction. In [NYC14], they observed similar results for fooling images. In their experiment, adversarial training with fooling images, deep neural network models are more robust against a limited set of fooling images. However they can still be fooled by other fooling images easily.

3.5 Conclusion

We show in this chapter that the binary classifier is a simple yet effective and robust way to separating adversarial from the original clean images. Its advantage over adversarial training and knowledge distillation is that it serves as a preprocessing step without assumptions about the model it protects. Besides, it can be readily deployed without any modification of the underlying systems. However, as we empirically showed in the experiment, the binary classifier approach, defensive training and distillation all suffer from the generalization limitation. For future work, we plan to extend our current work in two directions. First,

we want to investigate the disparity between different adversarial crafting methods and its effect on the generated adversarial space. Second, we will also carefully examine the cause of adversarial samples since intuitively the linear hypothesis does not seem right to us.

Chapter 4

Generate Text Adversarial Samples

4.1 Introduction

The adversarial images have been extensively studied. Many adversarial generating methods have been proposed in the literature, e.g, fast gradient method (FGM) [GSS14], Jacobian-based saliency map approach (JSMA) [Pap+15b], DeepFool [MFF15], CW [CW16], etc. Many theoretical explanation of adversarial samples also focused on image data and architectures [Pec+17; GSS14]. Some work have expanded the study to other domains, e.g, speech-to-text [CW18], neural translation [ZDS17], reinforcement learning [Lin+17], etc. These extended work will give us a more thorough understanding of the adversarial samples. To this end, we propose a simple yet effective framework to adapt the adversarial methods for images to generating adversarial texts. Specifically, we focus on adversarial samples for text classification models. There are two major difficulties to generate adversarial texts:

1. The input space is discrete. As a result, it is unclear how to (iteratively) accumulate small noise to perturb the input. Working with Image domain is easier since we usually normalize the input to a continuous domain $[0, 1]$.

2. The text quality measurement and control is intricate in itself. It is a very subjective matter. For example, let's compare the Master Yoda-style way of speaking, *Much to learn, you still have*, with the mundane-style, *You still have much to learn.* Which is better? Which gets a high score? Star Wars fans will definitely favor the Yoda-style, although both sentences successfully convey exactly the same meaning.

To resolve the first problem, we propose a general framework in which we generate adversarial texts via slightly modified methods borrowed from image domain. We first search for adversarials in the text embedding space (e.g., word-level embedding [Mik+13b], character-level embedding [Kim+15]), and then reconstruct the adversarial texts with nearest neighbor search. The second problem is open-ended, we employ two metrics to quantify the results, i.e., the Word Mover's Distance (WMD) [Kus+15] and change ratio (the number of words changed). In our experiments, they serve their purpose well at a rather coarse level. These two metrics, however, does not perform consistently when two text pieces are about the same quality (e.g., the aforementioned Yoda-style and mundane-style). The text quality is controlled empirically by the noise level in our experiments.

The contribution of our work lies in two-folds:

1. We propose a general framework to generate adversarial texts. Any of the existing adversarial methods may be adapted to generate adversarial texts under our framework.

2. We empirically compare the word-level and character-level adversarial texts, e.g., transferability, text quality, etc.

This chapter is organized as follows. we survey recent work on generating adversarial texts in Section 4.2. Our adversarial text framework is detailed in Section 4.3. We thoroughly evaluate our framework and compare with Hotflip [Ebr+17] on various text benchmarks and report the results in Section 4.4. We then conclude this chapter and provide directions for future work in Section 4.5.

4.2 Related Work

4.2.1 Text-space Methods

This class of methods perturbs the input texts directly. One disadvantage is that the computation cost is usually very high. To perturb the input texts directly, two decisions need to be made:

1. *What to change.* Generally speaking, the words that have more influence on the result should be altered first. Similar to JSMA, [Lia+17; SM17] compute importance score for each word based on ∇L or ∇f. In [JL17], the author manually construct fake facts around the sentence that contains the answer. [Lia+17] alters the input sentence in a brutal-force way, where each word is altered in sequence until an adversarial sample is found or a threshold on the maximum number of words to change is reached.

2. *Change to what.* Typos usually achieve good results, as shown in [SM17; Lia+17]. The disadvantage of typos is that they are relatively easier to be corrected by the auto spelling correction applications, e.g., Grammerly. Replacing with synonyms and antonyms (e.g., from Thesaurus) is also a good choice [Lia+17; SM17]. [Lia+17] uses semantically related words as potential replacements. As text embeddings [Mik+13b] have been shown to preserve semantic relations among words, the semantically related words can be approximated by nearest neighbor search in the embedding space.

4.2.2 Transformed-space Methods

This class of methods first map text inputs to a smooth space and search for potential adversarial samples in the smooth space via methods borrowed from adversarial images generation. Then the adversarial texts are reconstructed and further verified in the original text space. Usually some portion of the reconstructed texts are unsuccessful adversarial samples and are filtered out.

[ZDS17] employs an autoencoder to map between the input text and a Gaussian noise space. The decoder is a generator (i.e., GAN [Goo+14]), while the encoder is an MLP (called inverter in the paper). They search in the noise space with random walk. However, the disadvantage is that they do not have an explicit control of the quality of the adversarial samples. As we have see in [ZDS17], the generated adversarial images on complex dataset usually have large visual changes. Similarly, another generator-based method is proposed in [Won17] where the whole network is trained with REINFORCE [Wil92] algorithms.

In [Lia+17], the authors attempt FGM directly on character-level convolution networks [ZZL15]. Although the labels of the text pieces are successfully altered, the texts are changed to basically random stream of characters which is beyond understanding.

A highly related work is also report in [Ebr+17] where the authors conduct character-level and word-level attack based on gradients. The difference is that we use nearest neighbor search to reconstruct the adversarial sentences, while they search for adversarial candidates directly in the text space. Furthermore, the word-level adversarial texts were not very successfully in [Ebr+17]. Moreover, in our experiment, we also find that Hotflip has label leaking problem [KGB16a] as is the vanilla FGSM where the true labels are used to generate the adversarial texts. We fix this problem as suggested in [KGB16a] by using the predicted labels instead of the true ones to generate adversarial texts.

4.3 Adversarial Text Framework

In this section, we propose a general framework that generates adversarial texts with adapted methods for adversarial images. Our framework focuses on *replacing* words.

4.3.1 System Overview

Our system consists mainly of three parts, the embedding part, the adversarial generator, and the reverse embedding part. The embedding part maps raw input texts into a

continuous space. The reverse embedding part maps the perturbed embedding vectors back to texts.

4.3.2 Discrete Input Space

The first problem we need to resolve is how we can accumulate small noise to change the input. The idea comes from the observation that the first layer for most text models is the embedding layer. Thus, instead of working on the raw input texts, we first search for adversarials in the embedding space via gradient-based methods, and then reconstruct the adversarial sentences. Searching for adversarials in the embedding space is similar in principle to searching for adversarial images. However, the generated noisy embedding vectors usually do not correspond to any tokens in the text space. To construct the adversarial texts, we align each embedding to its nearest one via (approximate) nearest neighbor search. This reconstructing process can be seen as a strong *denoising* process. With appropriate noise scale, we would expect most of the words/characters remain unchanged, while only few are replaced. This framework builds upon the following observations.

1. When generating adversarial samples, the input features (e.g., pixels, words, characters) that are relatively more important for the final predictions will receive more noise, while others less noise. This property is intuitively illustrated in Figure 1.1, where usually only a subset of the pixels are perturbed. Despite that most pixels are perturbed in FGM, only a few pixels receive very large noise.

2. The embedded word vectors preserve the subtle semantic relationships among words [Mik+13b; Mik+13a]. For example, `vec"clothing"` is closer to `vec("shirt")` as `vec("dish")` to `vec("bowl")`, while `vec("clothing")` is farther way from `vec("dish")` or `vec("bowl")`, in the sense of p-norm, since they are not semantically close [MYZ13]. This property assures that it is more likely to replace the victim words with a semantically related one rather than a random one.

4.3.3 Word Mover's Distance (WMD)

For the second problem, we use two metrics to quantify the adversarial texts quality, the Word Mover's Distance (WMD) [Kus+15] and the change ratio (i.e., the number of words changed divided by the maximum sequence length). WMD measures the dissimilarity between two text documents as the minimum amount of distance that the embedded words of one document need to *travel* to reach the embedded words of another document. WMD can be considered as a special case of Earth Mover's Distance (EMD) [RTG00]. Intuitively, it quantifies the semantic similarity between two text bodies. A lower WMD score means a better adversarial samples. As we will see in our experiments, WMD is only good as a coarse-level metric.

4.4 Experiment

We evaluate our framework on three text classification problems. Section 4.4.1 details on the data preprocessing. The adversarial methods we use in our experiment are (FGM) [GSS14] and DeepFool [MFF15]. We report the model accuracy on clean sample as well as adversarial texts.

Detailed discussion follow each experiment results. Only a few examples of generated adversarial texts are shown in this paper due to the space constraint. More samples of adversarial texts under different parameter settings and the code to reproduce the experiment are available online[1].

Computation-wise, the bottleneck in our framework is the nearest neighbor search. Word vector spaces, such as GloVe [PSM14], usually have millions or billions of tokens embedded in very high dimensions. The nearest neighbor search is slow. Instead, we employ the approximate nearest neighbor (ANN) technique in our experiment. The ANN implementation

[1] `https://github.com/gongzhitaao/adversarial-text`

which we use in our experiment is Approximate Nearest Neighbors Oh Yeah (`annoy`)[2], which is well integrated into `gensim` [ŘS10] package.

4.4.1 Dataset

We use three text datasets in our experiments. The datasets are summarized in Table 4.1. The last column shows our target model accuracy on the clean test data.

Dataset	Labels	Training	Testing	Seq. Len.	Word Len.	Accuracy
IMDB	2	25000	25000	300	20	0.8787
Reuters-2	2	3300	1438	100	20	0.9854
Reuters-5	5	1735	585	100	20	0.8701

Table 4.1: Dataset Summary

IMDB Movie Reviews

This is a dataset for binary sentiment classification [Maa+11]. It contains a set of 25,000 highly polar (positive or negative) movie reviews for training, and 25,000 for testing. No special preprocessing is used for this dataset except that we truncate/pad all the sentences to a fixed maximum length.

Reuters

This is a dataset of 11,228 newswires from Reuters, labeled over 90 topics. We load this dataset through the NLTK [BKL09] package. The raw Reuters dataset is highly unbalanced. Some categories contain over a thousand samples, while others may contain only a few. The problem with such highly unbalanced data is that the texts that belong to under-populated categories are almost always get classified incorrectly. Even though our model may still achieve high accuracy with 90 labels, it would be meaningless to include these under-populated categories in the experiment since we are mainly interested in perturbation

[2] `https://github.com/spotify/annoy`

of those samples that are already being classified correctly. Keras[3] uses 46 categories out of 90. However, the 46 categories are still highly unbalanced. In our experiment, we preprocess Reuters and extract two datasets from it, i.e., Reuters-2 and Reuters-5.

1. Reuters-2 It contains two most populous categories, i.e., `acq` and `earn`. The `acq` category contains 1650 training samples and 719 test samples. Over 71% sentences in the `acq` category have less than 160 tokens. The `earn` category contains 2877 training samples and 1087 test samples. Over 83% sentences in `earn` category have less then 160 tokens. In order to balance the two categories, for `earn`, we use 1650 samples out of 2877 for training, and 719 for testing. The maximum sentence length of this binary classification dataset is set to 100.

2. Reuters-5 It contains five categories, i.e., `crude`, `grain`, `interest`, `money-fx` and `trade`. Similar to Reuters-2, we balance the five categories by using 347 examples (the size of `interest` categories) for each category during training, and 117 each for testing. The maximum sentence length is set to 100.

4.4.2 Embedding

Our framework relies heavily on the *size* and *quality* of the embedding space. More semantic alternatives would be helpful to improve the quality of generated adversarial texts. As a result, we use the GloVe [PSM14] pre-trained embedding in our experiment. Specifically, we use the largest GloVe embedding, `glove.840B.300d`, which embeds 840 billion tokens (approximately 2.2 million cased vocabularies) into a vector space of 300 dimensions. The value range of the word vectors are roughly $(-5.161, 5.0408)$.

[3] `https://keras.io/`

4.4.3 Model

In this work, we tested two commonly used architectures for sequence classification problem. The first one is a word-level convolution network [Kim14] (as shown in Figure 4.1). This architecture differs from the image models in two aspects: (i) an embedding layer is added right after the input to map the word indices to their corresponding vector representations, and (ii) the pooling layers are global max-pooling.

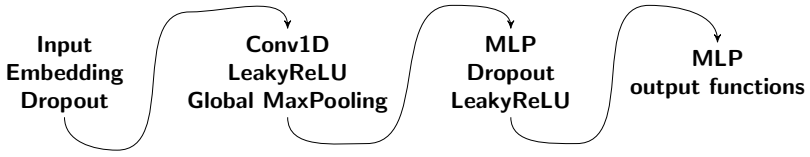


Figure 4.1: Word-level CNN model for text classification. (Image credit: [Gon+18])

The other one is a character-aware model [Kim+15]. The first layer is an embedding layer, followed by parallel convolution layers of *different* filter sizes, which all go through a global max-pooling layer. The outputs of different pooling layers are then concatenated before going through a highway layers [SGS15] and LSTMs. Please refer to [Kim+15] for a detailed discussion about the architecture.

The detailed parameter settings are available in our released code. Note that for models trained on binary classification tasks, DeepFool assumes the output in the range $[-1, 1]$, instead of $[0, 1]$. Thus we have two slightly different models for each of the binary classification task (IMDB and Reuters-2), the one with `sigmoid` output, and the other with `tanh`. The models with `tahn` output are trained with Adam [KB14] by minimizing a root mean squared error (RMSE), while all the other models are trained with Adam by minimizing a cross-entropy loss. Despite the small difference in architecture, `sigmoid`- and `tanh`-models on the same task have almost identical accuracy. As a result, in Table 4.1, we report only one result for IMDB and Reuters-2. In the following sessions, we refer to the word-level model as `WordCNN`, the character-level model as `CharLSTM`. Wherever necessary, the binary

classification model with `sigmoid` output is suffixed with `-sigm`, e.g, `WordCNN-sigm`, the one with `tanh` output is suffixed with `-tanh`, e.g., `WordCNN-tanh`.

4.4.4 Effectiveness and Quality Trade-off

ϵ	WMD	N	Text Piece
20	0.1332	1/100	HBO URGES ~~SHAREHOLDERS~~ DIVIDEND AGAINST ANDOVER HBO and Co said it sent a letter of strongly urging shareholders not to sign any proxy cards sent by Andover Group . ON March 30 , Andover Group , a two-man general partnership which owns about seven pct of HBO 's stock , filed preliminary proxy materials with the Securities and Exchange Commission seeking to nominate an alternative slate of directors at the company 's April 30 annual meeting . Andover had expressed an interest to acquire the company in September 1986 but HBO has never received an offer from them , it said
30	0.1629	2/100	HBO ~~URGES~~ GAINS ~~SHAREHOLDERS~~ DIVIDEND AGAINST ANDOVER HBO and Co said it sent a letter of strongly urging shareholders not to sign any proxy cards sent by Andover Group . ON March 30 , Andover Group , a two-man general partnership which owns about seven pct of HBO 's stock , filed preliminary proxy materials with the Securities and Exchange Commission seeking to nominate an alternative slate of directors at the company 's April 30 annual meeting . Andover had expressed an interest to acquire the company in September 1986 but HBO has never received an offer from them , it said
40	0.2013	5/100	~~HBO~~ MTV ~~URGES~~ INCREASE ~~SHAREHOLDERS~~ DIVIDEND AGAINST ANDOVER HBO and Co said it sent a letter of strongly urging shareholders not to sign any proxy cards sent by Andover Group . ON March 30 , Andover Group , a two-man general partnership which owns about seven pct of HBO 's stock , filed preliminary proxy materials with the Securities and Exchange Commission seeking to nominate an alternative slate of directors at the company 's April 30 annual meeting . Andover had expressed an interest to acquire the company in ~~September~~ December ~~1986~~ 1987 but HBO has never received an offer from them , it said
50	0.1998	6/100	~~HBO~~ MTV ~~URGES~~ INCREASE ~~SHAREHOLDERS~~ RESIDUAL AGAINST ANDOVER HBO and Co said it sent a letter of strongly urging shareholders not to sign any proxy cards sent by Andover Group . ON March 30 , Andover Group , a two-man general partnership which owns about seven pct of HBO 's stock , filed preliminary proxy materials ~~with~~ and the Securities and Exchange Commission seeking to nominate an alternative slate of directors at the company 's April 30 annual meeting . Andover had expressed an interest to acquire the company in ~~September~~ December ~~1986~~ 1987 but HBO has never received an offer from them , it said

Figure 4.2: Adversarial texts generated with Deepfool with different noise scale on word-level model. (Image credit: [Gon+18])

If the model's accuracy on the adversarial texts are lower, then we say the adversarial texts are more *effective*. The quality of the adversarial texts refers to grammar and syntactic correctness of the text piece. We employ several intuitive criteria to measure the quality of the adversarial texts, i.e., the number of words changed (N) and the Word Mover's Distance (WMD). The number-of-words measurement makes sense in our settings since our framework will only replace words, no addition and deletion. The trade-off between the effectiveness and quality of the adversarial texts is controlled by the noise level. As expected, large noise level would generate more effective adversarial samples. However, the text quality will also degrade with larger noise.

Figure 4.3 shows the trade-off for FGSM method. As we can see, the quality of adversarial texts generated by FGSM deteriorates quickly as we increase the noise level. Albeit It

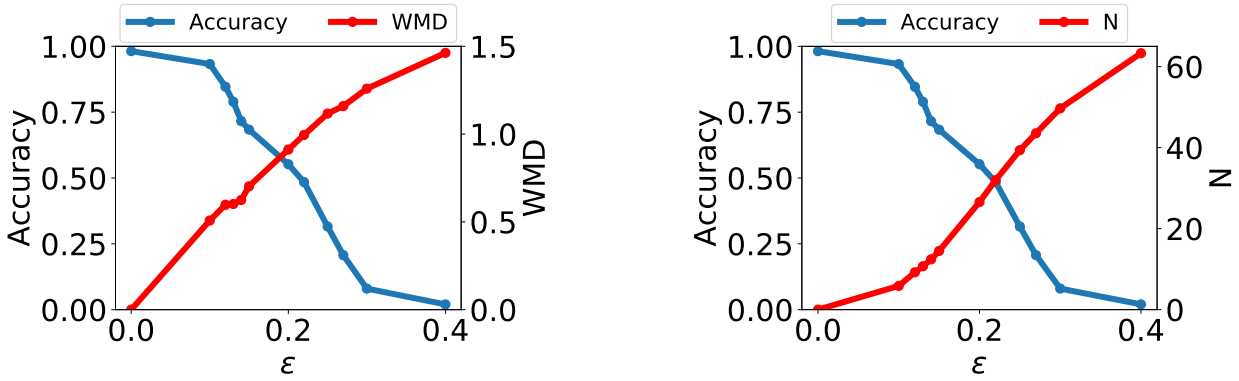


Figure 4.3: Word-level model's accuracy with varying FGSM noise level. The WMD and N (number of words changed) empirically show the quality of the adversarial texts. (Image credit: [Gon+18])

becomes more effective toward the target model. Especially, the number of words changed grows rapidly. Figure 4.4 shows the trade-off for DeepFool method. It follows a similar trend as FGSM in general. However, we can see that DeepFool generates much better adversarial texts then FGSM when they are similar in effectiveness. This is similar in the case of adversarial images. FGSM tends to add noise to all the dimension of the input, thus with larger noise, we would expect most words are changed. On the other hand, DeepFool usually changes only a small subset of the input dimension. Even with a larger noise, most words remain unperturbed.

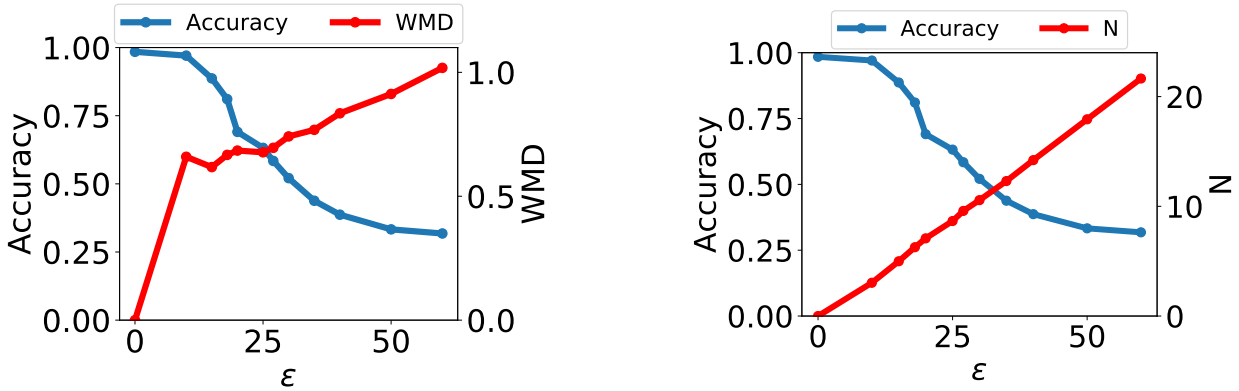
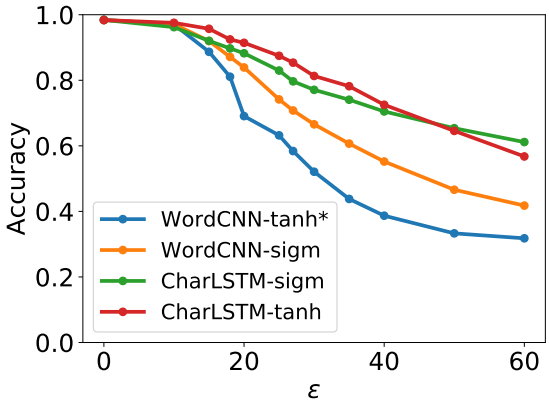


Figure 4.4: Word-level model's accuracy with varying DeepFool overshoot value. The WMD and N (number of words changed) empirically show the quality of the adversarial texts. (Image credit: [Gon+18])

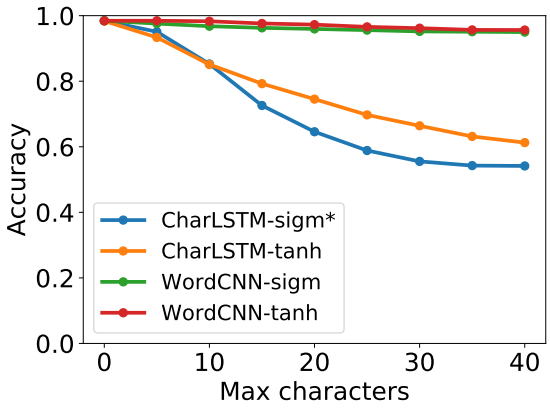
The examples of adversarial texts generated via DeepFool at different noise level are shown in Figure 4.2. The WMD and number of words changed are also included to give an intuition about the correspondence between the measurements and the text quality.

4.4.5 Transferability

We test the transferability of adversarial texts generated on word-level models and character-level models, respectively. In our experiments, word-level adversarial texts show very good transferability, even to character-level models. However character-level adversarial texts do not transfer well to word-level models.



(a) Transferability of adversarial texts generated via DeepFool on word-level. The WordCNN-tanh is the model used to generated the adversarial texts.



(b) Transferability of adversarial texts generated via Hotflip on character-level. The CharLSTM-sigm is the model used to generated the adversarial texts. (Image credit: [Gon+18])

Figure 4.5a shows the transferability of word-level adversarial texts generated in our framework via DeepFool. The adversarial texts are generated on WordCNN-tanh model. The adversarial texts transfer better to WordCNN-sigm which shares a similar structure as WordCNN-tanh except for the output function. Figure 4.5b shows the transferability of character-level adversarial texts generated via Hotflip [Ebr+17]. The character-level adversarial texts only show transferability to character-level models, but not to word-level models. The main reason is that the changes to character-level adversarial texts are mainly within

words. In most cases, the perturbed words will be replaced by unknown word placeholder (e.g., `<unk>` in our experiments) they are rarely legit. Thus the character-level adversarial texts basically degrade to unknown-word adversarials for word-level models. As expected, replacing only a few words with `<unk>` is not enough to fool the word-level model.

4.4.6 Defense

The defense for character-level adversarial texts are relatively easy, most of the errors can be easily corrected by auto-spelling applications, e.g., Grammerly[4], Bing Spell Check API. The incorrect spellings are easy to detect and recover, e.g., *sontware* is successfully corrected to *software*. However, if the character is replaced by punctuation characters, the word will be not corrected, e.g., *qu{kly* is not recognized and correct.

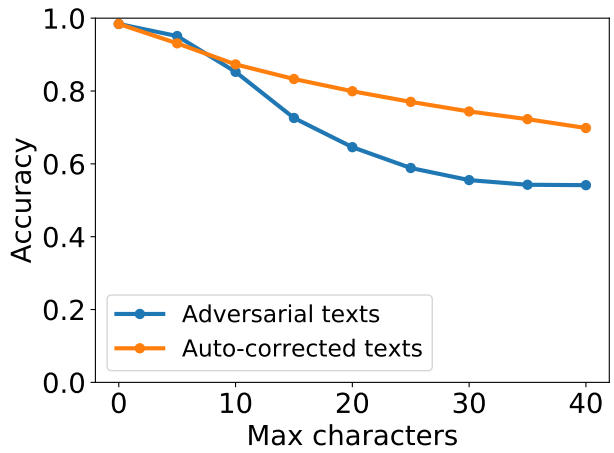


Figure 4.6: Defense against character-level adversarials. (Image credit: [Gon+18])

When generating the character-level adversarial texts, we want to change as few characters as possible so that the resulting adversarial texts do not degrade into garbage. However, the fewer characters we change, the easier they are corrected by auto-spelling applications.

[4] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

4.4.7 Results on Word-Level Model

The noise scale (ϵ in Table 4.2) influences the effectiveness of adversarial methods, as well as the the quality of generated adversarial sentences. The model accuracy under different noise scales are summarized in Table 4.2.

Table 4.2: Word-level CNN accuracy under different parameter settings. ϵ is the noise scaling factor.

Method	Dataset		Accuracy			
FGSM		ϵ	0.40	0.35	0.30	0.25
	IMDB		0.1334	0.1990	0.4074	0.6770
	Reuters-2		0.6495	0.7928	0.9110	0.9680
	Reuters-5		0.5880	0.7162	0.7949	0.8462
FGVM		ϵ	15	30	50	100
	IMDB		0.8538	0.8354	0.8207	0.7964
	Reuters-2		0.7990	0.7538	0.7156	0.6523
	Reuters-5		0.7983	0.6872	0.6085	0.5111
DeepFool		ϵ	20	30	40	50
	IMDB		0.8298	0.7225	0.6678	0.6416
	Reuters-2		0.6766	0.5236	0.4910	0.4715
	Reuters-5		0.4034	0.2222	0.1641	0.1402

Figure 4.2 shows one example of adversarial texts generated via DeepFool [MFF15] in our framework with different noise levels. The ϵ in the first column is the noise level (i.e., the overshoot value in DeepFool algorithm), the second column the word mover's distance value, the third the number of word(s) changed. All the adversarial texts are generated from the same sample, the only difference is the noise level. As we could see, as we increase the noise level, more words are changed as expected. Furthermore, the WMD value increases as well. Essentially, the noise level controls the balance between the quality and effectiveness of the generated adversarial texts.

In the adversarial text examples, the ~~original~~ words their corresponding adversarial words they are changed into are highlighted respectively to aid reading.

We evaluate two versions of FGM, i.e., fast gradient sign method (FGSM) and fast gradient value method (FGVM). Their example results are shown in Figure 4.7 and Figure 4.8, respectively. With appropriate noise level, we can change only a few words to alter the label of the whole text piece.

WMD	N	Text Piece
0.0382	1	Lifeforce starts in outer space where the HMS Churchill tracks Haley 's Comet & it 's equipment detects a 150 mile long alien spaceship in the head of the comet , unable to contact Earth because of interference Colonel Tom Carlsen (Steve Railsback) decides this is the one & only chance to investigate it . Going outside in spacesuits Carlsen & some of his crew enter the mysterious spaceship & find the remains of a bat like race of creatures & three perfect looking humanoids , two men (Chris Jagger & Bill Malin) & a beautiful woman (Mathilda May) all of whom they take aboard the Churchill . Thirty days later & back on Earth the Churchill is detected on radar , a rescue mission is sent up only to discover the spaceship is burnt out & all the crew supposedly dead . The rescue team do find the three humanoid aliens though & take them back to Earth where in a space research center in London they come back to life & start to literally suck the lifeforce out of human victims who then in turn need to do the same to stop themselves turning into dust , things look grim as the epidemic spreads throughout London ... This English production was produced by the notorious Menahem Golan & Yoram Globus who during the early 80 's were responsible for lots of cheap low budget action flicks under their production company Cannon usually starring Chuck Norris & they wanted to move into the big time & signed director Tobe Hooper up on a three film deal (which were this ~~Lifeforce~~ HEATR , the Invaders from Mars (1986) remake & The Texas Chainsaw Massacre 2 (1986) a sequel to Hooper
0.0465	2	This was very funny , even if it fell apart a little at the end . Does not go overboard with homage after to Hitchcock - Owen (Danny DeVito) was lucky he had " Strangers on A Train " playing at the local cinema , so the movie flat out tells you that that was the inspiration . DeVito is very funny but also a little sad . He has no friends and all he wants to do is ~~write~~ bother and have someone like his writing . His teacher , Billy Crystal , is going through some serious writers block of his own and his wife has stolen his book and made it her own success , which also has him frustrated a great ~~deal.Best~~ <unk> parts are the book proposal by Mr. Pinsky (" One Hundred Girls I 'd Like to Pork ") and all scenes with Anne Ramsey , who is so horrible that even Mother Theresa would have wanted to kill her , too !
0.0254	2	This show is my guilty pleasure all the way ! ! When I first tuned in to America 's Next Top Model , I expected to be bored ~~-~~ instead and to find it very very stupid . I did n't . This show is actually serious fun . I read on one of the other reviews that it makes you wonder if you have what it takes to be America 's Next top model . And it so does ! Who does n't love the glamour and excitement that come with being a model ? On ANTM you get to see what it 's REALLY like . And who does n't love hearing the girls bitch about each other and get ~~into~~ down fights ? Or enjoy wanting to throw something at that Janice lady ? Give this a chance . Do n't expect something intelligent or a show you can look to for a life lesson . Just enjoy it for what it is . Serious fun !

Figure 4.7: IMDB adversarial texts generated via FGSM on word-level model, $\epsilon = 0.08$. (Image credit: [Gon+18])

4.4.8 Results on Character-level Model

The results for character-level adversarials are more interesting. One example is shown in 4.9. For relatively small noise, we observe similar adversarial texts, i.e., only a few characters are changed while the whole sentence is still legit albeit its label is already different from the clean sample. On the other hand, if we tune up the noise level, the whole sentence is changed to another somewhat legit sentence, which is rather surprising. With a large noise, we would expect that the whole sentence turn into garbage, as observed in [Lia+17],

WMD	N	Text Piece
0.0338	1	Chris Nolan 's labyrinth like noir about voyeurism and identity is amazing from start to finish . A first film is as complex as " Memento " or " The Prestige " , though maybe a little harder to get a handle on . Still it ~~smacks~~ pathetically of originality and creative drive , and has a " twist " as intellectually challenging as it is realistic pulp . Few film makers have made as good of use of their editors and attention to narrative that Nolan has . The story is about a bored writer who likes to follow random strangers down the street , until he follows someone , whose noticed him following others , and has been following him in tern , from there the complexity escalates and identities begin to rearrange . More naturalistic and realist than Nolan 's later work but just as razor sharp .
0.0427	3	This movie was released originally as a soft ~~"~~ authority X " , apparently with the explicit sex deleted . Later , the producers " re-lented " (smelled money) and re-released it with the excised scenes restored (apparently only about 3 minutes) . I guess since Kris-tine was of age , it was held against her and her promising career came grinding to a halt . I guess ~~its~~ Its all in the timing (witness Pam Anderson 's career) – but Ronald Reagan was in charge during Kristine 's debacle (we had not heard about Nancy Reagan 's af-fairs) , Bill Clinton and Monica Lewinski were in full swing during Pam 's " coming out " ~~.The~~ <unk> sex is just icing on the cake , both version satisfy . This naughty musical is way above similar of others that were released at the same time .
0.0227	1	This thriller is one of the few (film) surprises I 've had in quite some time . Everybody - and I do mean EVERYBODY - I talked to when it was in the theatres said it was awful ... but then I got to thinking ... none of these people really understand ~~horror/thriller~~ action/thriller films or metal rock - so WHY DO I LISTEN TO THEM IN THE FIRST PLACE ? ? ? ? ? This film kicks ass - but ONLY if you are able to comprehend and enjoy this type of entertainment . In short (too late) - see it with an open mind and it might just open your mind . ... the soundtracks great too ...

Figure 4.8: IMDB adversarial texts generated via FGVM on word-level model with varying ϵ. (Image credit: [Gon+18])

ϵ	WMD	Text Piece
15	0.0362	UNION PLANTERS ~~ACQUISITIONS~~ ATSTUBITIONS APPROVED Union Planters Corp said it has received regulatory approvals for its previously-announced acquisitions of Borc Financial Corp and First Citizens Bank of Hohenwald , and approval of its acquisition of Merchants State Holding Co is expected within 10 days . All are to be completed during the second quarter of 1987 , it said .
30	1.8664	CHReSLER NON-PROFIT GROUP {{SST UNIT Chrysler Corp 's Chrysler Motors Corp said its Chrysler Training Corp non-profit or-ganization sold the name and assets of its Motech Auto Mechanic and Uody Shop Schools to O/E Corp of Troy , Mich . The sale price was not disclosed . Under the Internal Revenue Service code , proceeds from the sale of Motech must be donated to another tax-exempt nonprofit organization . Chrysler did not reveal the name of the group that received the proceeds .

Figure 4.9: Adversarial texts generated with Deepfool with different noise scale on character-level model. Both adversarial samples are generated from the same clean sample. The second adversarial sample is generated by adding a very large noise. (Image credit: [Gon+18])

albeit the resulting sentence does change to a different category. Our hypothesis is that the architecture of the character-level model plays an important role.

1. The different width of feature maps encode different levels of contextual information around each character.

2. The highway and LSTM layer mingles the contextual information together so that the noise follows a certain direction.

4.5 Conclusion

In this work, we proposed a framework to adapt image attacking methods to generate high-quality adversarial texts in an end-to-end fashion, without relying on any manually selected features. In this framework, instead of constructing adversarials directly in the raw text space, we first search for adversarial embeddings in the embedding space, and then reconstruct the adversarial texts via nearest neighbor search. We demonstrate the effectiveness of our method on three texts benchmark problems. In all experiments, our framework can successfully generate adversarial samples with only a few words changed. In addition, we also empirically demonstrate Word Mover's Distance (WMD) as a valid quality measurement for adversarial texts. In the future, we plan to extend our work in the following directions.

1. WMD is demonstrated to be a viable quality metric for the generated adversarial texts. We can employ the optimization and model attacking methods by minimizing the WMD.

2. We use a general embedding space in our experiments. A smaller embedding that is trained on the specific task may help to speed up the computation needed to reconstruct the texts.

3. If we plug-in an autoencoder (e.g., the sequence to sequence architecture) as the input to the classification model, we then can eliminate the reconstruction process which is the computation bottleneck in our current framework.

Chapter 5

Generate *Natural* Adversarial Images

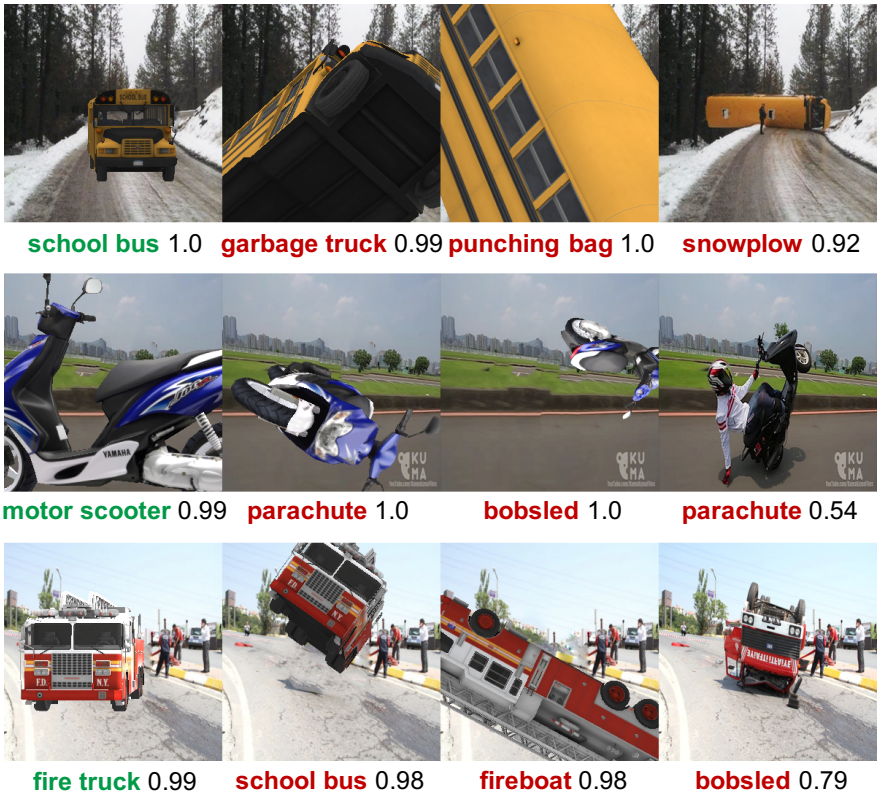


Figure 5.1: The Google Inception-v3 classifier [Sze+16] correctly labels the canonical poses of objects (a), but fails to recognize out-of-distribution images of objects in unusual poses (b-d), including real photographs retrieved from the Internet (d). The left 3 by 3 images (a-c) are found by our framework and rendered via a 3D renderer. Below each image are its top-1 predicted label and confidence score. (Image credit: [Alc+18])

5.1 Introduction

For real-world technologies, such as self-driving cars [Che+15], autonomous drones [GPG17], and search-and-rescue robots [Sam+18], the test distribution may be non-stationary, and

new observations will often be out-of-distribution (OoD), i.e., not from the training distribution [SLS+17].

However, machine learning (ML) models frequently assign wrong labels with high confidence to OoD examples, such as adversarial examples [Sze+13; NYC14] inputs specially crafted by an adversary to cause a target model to misbehave. But ML models are also vulnerable to *natural* OoD examples [Lam16; Gra18; Tia+18; Tim16]. For example, when a Tesla autopilot car failed to recognize a white truck against a bright-lit sky, an unusual view that might be OoD, it crashed into the truck, killing the driver [Tim16].

To understand such natural Type II classification errors, we searched for 6D poses (i.e., 3D translations and 3D rotations) of 3D objects that caused DNNs to misclassify.

Our results reveal that state-of-the-art image classifiers and object detectors trained on large-scale image datasets [Rus+15; Lin+14] misclassify most poses for many familiar training-set objects. For example, DNNs predict the front view of a school bus, an object in the ImageNet dataset [Rus+15], extremely well (Figure 5.1 (a)) but fail to recognize the same object when it is too close or flipped over, i.e., in poses that are OoD yet exist in the real world (Figure 5.1 (d)).

Addressing this type of OoD error is a non-trivial challenge. First, objects on roads may appear in an infinite variety of poses [Tim16; Gra18]. Second, these OoD poses come from known objects and should be assigned known labels rather than being rejected as unknown objects [HG17; Sch+13]. Moreover, a self-driving car needs to correctly estimate at least some attributes of an incoming, unknown object (instead of simply rejecting it) to handle the situation gracefully and minimize damage.

In this chapter, we propose a framework for finding OoD errors in computer vision models in which iterative optimization in the parameter space of a 3D renderer is used to estimate changes (e.g., in object geometry and appearance, lighting, background, or camera settings) that cause a target DNN to misbehave (Figure 5.2). With our framework, we generated unrestricted 6D poses of 3D objects and studied how DNNs respond to 3D translations and

3D rotations of objects. For our study, we built a dataset of 3D objects corresponding to 30 ImageNet classes relevant to the self-driving car application. All code and data for our experiments will be available at `https://github.com/airalcorn2/strike-with-a-pose`. In addition, we will release a simple GUI tool that allows users to generate their own adversarial poses of an object. Our main findings are:

1. ImageNet classifiers only correctly label 3.09% of the entire 6D pose space of a 3D object, and misclassify many generated adversarial examples (AXs) that are human-recognizable (Figure 5.1 (b-c)). A misclassification can be found via a change as small as 10.31°, 8.02°, and 9.17° to the yaw, pitch, and roll, respectively.

2. 99.9% and 99.4% of AXs generated against Inception-v3 transfer to the AlexNet and ResNet-50 image classifiers, respectively, and 75.5% transfer to the YOLOv3 object detector.

3. Training on adversarial poses generated by the 30 objects (in addition to the original ImageNet data) did not help DNNs generalize well to held-out objects in the same class.

In sum, our work shows that state-of-the-art DNNs perform *image classification* well but are still far from true *object recognition*. While it might be possible to improve DNN robustness through adversarial training with many more 3D objects, we hypothesize that future ML models capable of visual reasoning may instead benefit from strong 3D geometry priors.

This chapter is organized as follows. We provide our framework details in Section 5.2. The experiments and discussion are provided in Section 5.3. We also provide details on our methods against the adversarial training defense in Section 5.4. Related work are discussed in Section 5.5. We then conclude this chapter in Section 5.6.

5.2 Framework

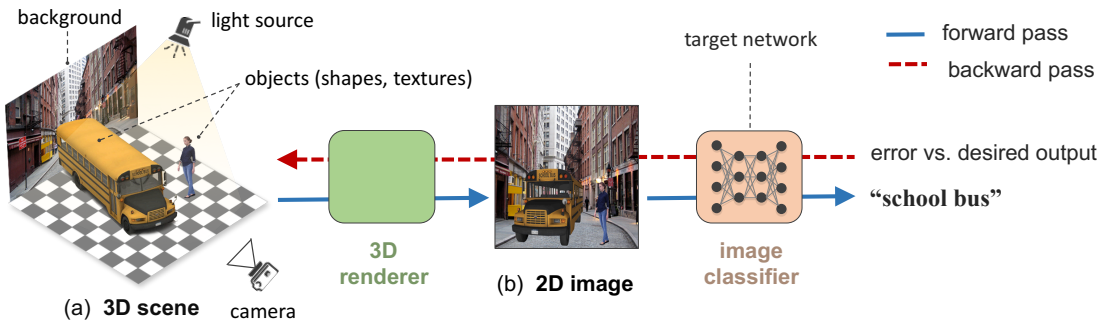


Figure 5.2: To test a target DNN, we build a 3D scene (a) that consists of 3D objects (here, a school bus and a pedestrian), lighting, a background scene, and camera parameters. Our 3D renderer renders the scene into a 2D image, which the image classifier labels `school bus`. We can estimate the pose changes of the school bus that cause the classifier to misclassify by (1) approximating gradients via finite differences; or (2) back-propagating (red dashed line dashed line) through a differentiable renderer. (Image credit: [Alc+18])

5.2.1 Problem formulation

Let f be an image classifier that maps an image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ onto a softmax probability distribution over 1000 output classes [Sze+16].

Let R be a 3D renderer that takes as input a set of parameters ϕ and outputs a render, i.e., a 2D image $R(\phi) \in \mathbb{R}^{H \times W \times C}$ (see Figure 5.2). Typically, ϕ is factored into mesh vertices V, texture images T, a background image B, camera parameters C, and lighting parameters L, i.e., $\phi = \{V, T, B, C, L\}$ [KUH18]. To change the 6D pose of a given 3D object, we apply a set of 3D rotations and 3D translations, parameterized by $\mathbf{w} \in \mathbb{R}^6$, to the original vertices V, yielding a new set of vertices V^*. Here, we wish to estimate only the pose transformation parameters $\mathbf{w}$ (while keeping all parameters in ϕ fixed) such that the rendered image $R(\mathbf{w}; \phi)$ causes the classifier f to assign the highest probability (among all outputs) to an incorrect target output at index t. Formally, we attempt to solve the below optimization problem:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}}(f_t(R(\mathbf{w}; \phi))) \tag{5.1}$$

In practice, we minimize the cross-entropy loss $\mathcal{L}$ for the target class. Equation 5.1 may be solved efficiently via backpropagation if both f and R are differentiable, i.e., we are able to compute $\partial\mathcal{L}/\partial\mathbf{w}$. However, standard 3D renderers, e.g., OpenGL [Woo+99], typically include many non-differentiable operations and cannot be inverted [MS15]. Therefore, we attempted two approaches:

1. harnessing a recently proposed differentiable renderer and performing gradient descent using its analytical gradients; and

2. harnessing a non-differentiable renderer and approximating the gradient via finite differences.

We will next describe the target classifier in Section 5.2.2. Section 5.2.3 provides details on the 3D renderers we are using. Our own dataset of 3D objects are introduced in Section 5.2.4. Then we discuss the optimization methods in Section 5.2.5.

5.2.2 Classification Networks

We chose the well-known, pre-trained Google Inception-v3 [Sze+16] DNN from the PyTorch model zoo [PyT18] as the main image classifier for our study (the default DNN if not otherwise stated). The DNN has a 77.45% top-1 accuracy on the ImageNet ILSVRC 2012 dataset [Rus+15] of 1.2 million images corresponding to 1,000 categories.

5.2.3 3D Renderers

Non-differentiable Renderer (NR)

We chose ModernGL [Dom19] as our non-differentiable renderer. ModernGL is a simple Python interface for the widely used OpenGL graphics engine. ModernGL supports fast, GPU-accelerated rendering. This renderer is referred as NR hereafter.

Differentiable Renderer (DR)

To enable backpropagation through the non-differentiable rasterization process, [KUH18] replaced the discrete pixel color sampling step with a linear interpolation sampling scheme that admits non-zero gradients. While the approximation enables gradients to flow from the output image back to the renderer parameters ϕ, the render quality is lower than that of our non-differentiable renderer (see Figure 5.6 for a comparison). This renderer is referred as DR hereafter.

5.2.4 3D Object Dataset

Construction

Our main dataset consists of 30 unique 3D object models (purchased from many 3D model marketplaces) corresponding to 30 ImageNet classes relevant to a traffic environment (Figure 5.7). The 30 classes include 20 vehicles (e.g., `school bus` and `cab`) and 10 street-related items (e.g., `traffic light`). See Section 5.7.1 for more details.

Each 3D object is represented as a mesh, i.e., a list of triangular faces, each defined by three vertices [MS15]. The 30 meshes have on average 9,908 triangles (Table 5.4). To maximize the realism of the rendered images, we used only 3D models that have high-quality 2D image textures. We did not choose 3D models from public datasets, e.g., ObjectNet3D [Xia+16], because most of them do not have high-quality image textures. That is, the renders of such models may be correctly classified by DNNs but still have poor realism.

Evaluation

We recognize that a reality gap will often exist between a render and a real photo. Therefore, we rigorously evaluated our renders to make sure the reality gap was acceptable for our study. From ∼100 initially-purchased 3D models, we selected the 30 highest-quality

models using the evaluation method below. First, we quantitatively evaluated DNN predictions on the renders. For each object, we sampled 36 unique views (common in ImageNet) evenly divided into three sets. For each set, we set the object at the origin, the up direction to $(0, 1, 0)$, and the camera position to $(0, 0, -z)$ where $z = \{4, 6, 8\}$. We sampled 12 views per set by starting the object at a 10° yaw and generating a render at every 30° yaw-rotation. Across all objects and all renders, the Inception-v3 top-1 accuracy was 83.23% (compared to 77.45% on ImageNet images [Sze+16]) with a mean top-1 confidence score of 0.78 (Table 5.5). See Section 5.7.1 for more details. Second, we qualitatively evaluated the renders by comparing them to real photos. We produced 56 (real photo, render) pairs via three steps:

1. we retrieved real photos of an object (e.g., a car) from the Internet;
2. we replaced the object with matching background content in Adobe Photoshop; and
3. we manually rendered the 3D object on the background such that its pose closely matched that in the reference photo.

Figure 5.8 shows example (real photo, render) pairs. While discrepancies can be spotted in our side-by-side comparisons, we found that most of the renders passed our human visual Turing test if presented alone.

Background Images

Previous studies have shown that image classifiers may be able to correctly label an image when foreground objects are removed (i.e., based on only the background content) [ZXY16].

Because the purpose of our study was to understand how DNNs recognize an object itself, a non-empty background would have hindered our interpretation of the results. Therefore, we rendered all images against a plain background with RGB values of $(0.485, 0.456, 0.406)$, i.e., the mean pixel of ImageNet images. Note that the presence of a non-empty background

should not alter our main qualitative findings in this paper, adversarial poses can be easily found against real background photos (Figure 5.1).

5.2.5 Methods

We will describe the common pose transformations (Section 5.2.5) used in the main experiments. We were able to experiment with non-gradient methods because: (i) the pose transformation space $\mathbb{R}^6$ that we optimize in is fairly low-dimensional; and (ii) although the NR is non-differentiable, its rendering speed is several orders of magnitude faster than that of DR. In addition, our preliminary results showed that the objective function considered in Equation 5.1 is highly non-convex (see Figure 5.5), therefore, it is interesting to compare random search vs. gradient descent using finite-difference (FD), approximated gradients vs. gradient descent using the DR gradients.

Pose Transformations

We used standard computer graphics transformation matrices to change the pose of 3D objects [MS15]. Specifically, to rotate an object with geometry defined by a set of vertices $V = \{v_i\}$, we applied the linear transformations in Equation 5.2 to each vertex $v_i \in \mathbb{R}^3$:

$$v_i^R = R_y R_p R_r v_i \tag{5.2}$$

where R_y, R_p, and R_r are the 3×3 rotation matrices for yaw, pitch, and roll, respectively (the matrices can be found in Section 5.7.5).

We then translated the rotated object by adding a vector $T = [x_\delta \; y_\delta \; z_\delta]^\top$ to each vertex:

$$v_i^{R,T} = T + v_i^R$$

In all experiments, the center $c \in \mathbb{R}^3$ of the object was constrained to be inside a sub-volume of the camera viewing frustum. That is, the x-, y-, and z-coordinates of c were within $[-s, s]$, $[-s, s]$, and $[-28, 0]$, respectively, with s being the maximum value that would keep c within the camera frame. Specifically, s is defined as:

$$s = d \cdot \tan \theta_v \tag{5.3}$$

where θ_v is one half the camera's angle of view (i.e., 8.213° in our experiments) and d is the absolute value of the difference between the camera's z-coordinate and z_δ.

Random Search

In reinforcement learning problems, random search (RS) can be surprisingly effective compared to more sophisticated methods [Suc+17]. For our RS procedure, instead of iteratively following some approximated gradient to solve the optimization problem in Equation 5.1, we simply randomly selected a new pose in each iteration. The rotation angles for the matrices in Equation 5.2 were uniformly sampled from $(0, 2\pi)$. x_δ, y_δ, and z_δ were also uniformly sampled from the ranges defined in Section 5.2.5.

z_δ-constrained Random Search

Our preliminary RS results suggest the value of z_δ (which is a proxy for the object's size in the rendered image) has a large influence on a DNN's predictions. Based on this observation, we used a z_δ-constrained random search (ZRS) procedure both as an initializer for our gradient-based methods and as a naive performance baseline (for comparisons in Section 5.3.4). The ZRS procedure consisted of generating 10 random samples of $(x_\delta, y_\delta, \theta_y, \theta_p, \theta_r)$ at each of 30 evenly spaced z_δ from -28 to 0. When using ZRS for initialization, the parameter set with the maximum target probability was selected as the starting point. When using the procedure as an attack method, we first gathered the maximum

target probabilities for each z_δ, and then selected the best two z_δ to serve as the new range for RS.

Gradient descent with finite-difference

We calculated the first-order derivatives via finite central differences and performed vanilla gradient descent to iteratively minimize the cross-entropy loss $\mathcal{L}$ for a target class. That is, for each parameter $\mathbf{w}_i$, the partial derivative is approximated by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} = \frac{\mathcal{L}(\mathbf{w}_i + \frac{h}{2}) - \mathcal{L}(\mathbf{w}_i - \frac{h}{2})}{h} \tag{5.4}$$

Although we used an h of 0.001 for all parameters, a different step size can be used per parameter. Because radians have a circular topology (i.e., a rotation of 0 radians is the same as a rotation of 2π radians, 4π radians, etc.), we parameterized each rotation angle θ_i as $(\cos\theta_i, \sin\theta_i)$, a technique commonly used for pose estimation [OML05] and inverse kinematics [CL92], which maps the Cartesian plane to angles via the $atan2$ function. Therefore, we optimized in a space of $3 + 2 \times 3 = 9$ parameters. The approximate gradient $\nabla\mathcal{L}$ obtained from Equation (5.4) served as the gradient in our gradient descent. We used the vanilla gradient descent update rule:

$$\mathbf{w} \coloneqq \mathbf{w} - \gamma \nabla \mathcal{L}(\mathbf{w})$$

with a learning rate γ of 0.001 for all parameters and optimized for 100 steps (no other stopping criteria).

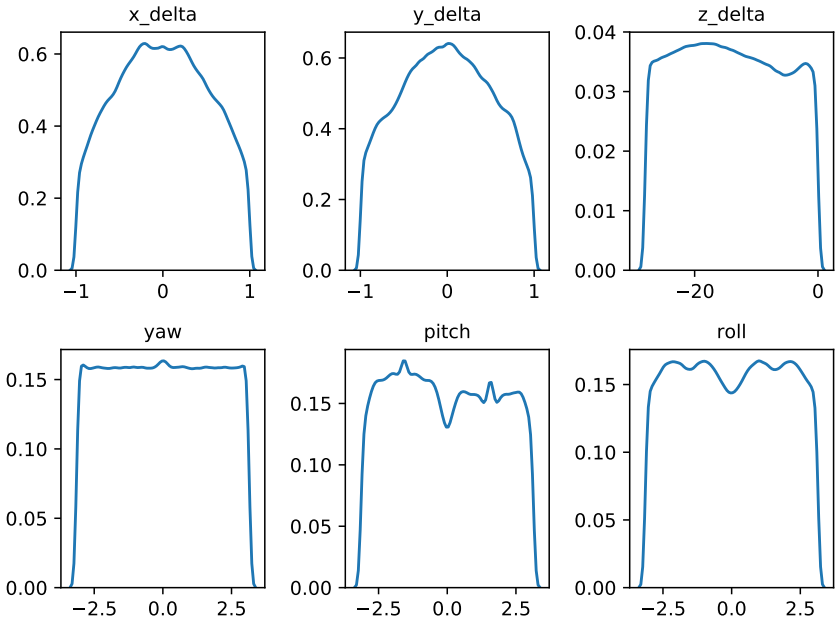


Figure 5.3: The distributions of individual pose parameters for high-confidence ($p \geq 0.7$) incorrect classifications. x_δ and y_δ have been normalized w.r.t. their corresponding s from Equation (5.3). (Image credit: [Alc+18])

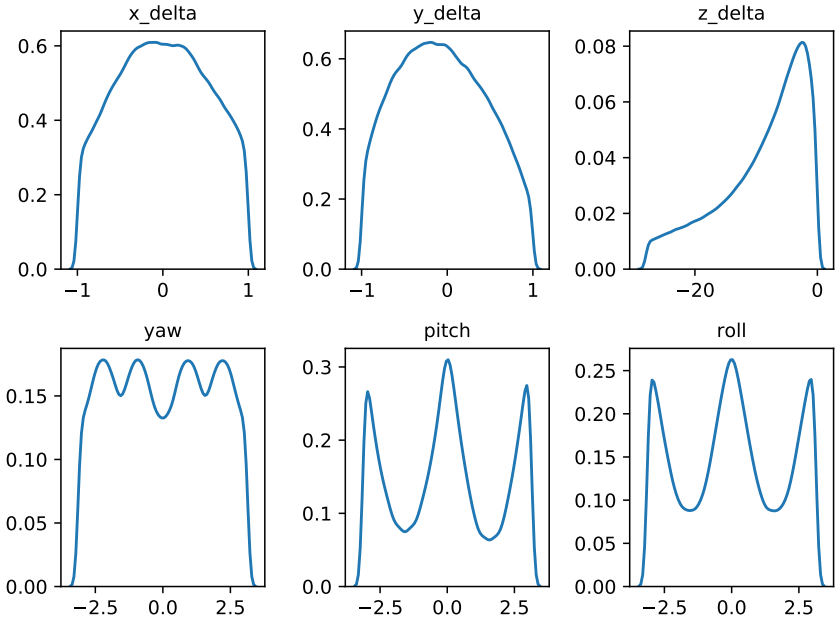


Figure 5.4: The distributions of individual pose parameters for high-confidence ($p \geq 0.7$) correct classifications obtained from the random sampling procedure described in Section 5.2.5. x_δ and y_δ are also normalized as well. (Image credit: [Alc+18])

5.3 Experiments and Results

5.3.1 Neural Networks Are Easily Confused by Object Rotations and Translations

Experiment

To test DNN robustness to object rotations and translations, we used RS to generate samples for every 3D object in our dataset. In addition, to explore the impact of lighting on DNN performance, we considered three different lighting settings: bright, medium, and dark (example renders in Figure 5.15). In all three settings, both the directional light and the ambient light were white in color, i.e., had RGB values of $1.0, 1.0, 1.0$), and the directional light was oriented at $(0, -1, 0)$ (i.e., pointing straight down). The directional light intensities and ambient light intensities were $(1.2, 1.6)$, $(0.4, 1.0)$, and $(0.2, 0.5)$ for the bright, medium, and dark settings, respectively. All other experiments used the medium lighting setting.

Misclassifications Uniformly Cover the Pose Space

For each object, we calculated the DNN accuracy (i.e., percent of correctly classified samples) across all three lighting settings (Table 5.8).

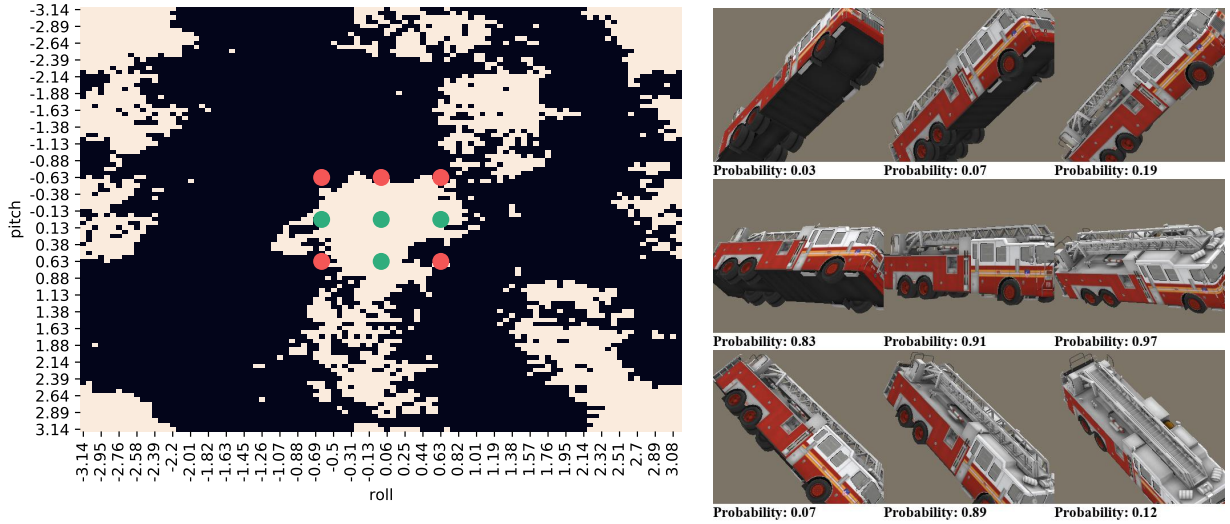
The DNN was wrong for the vast majority of samples, i.e., the median percent of correct classifications for all 30 objects was only 3.09%. Moreover, high-confidence misclassifications ($p \geq 0.7$) are largely uniformly distributed across every pose parameter (Figure 5.3), i.e., AXs can be found throughout the parameter landscape (see Figure 5.20 for examples). In contrast, correctly classified examples are highly multi-modal w.r.t. the rotation axis angles and heavily biased towards z_δ values that are closer to the camera (Figure 5.4).

An Object Can Be Misclassified As Many Different Labels

Previous research has shown that it is relatively easy to produce AXs corresponding to many different classes when optimizing input images [Sze+13] or 3D object textures [Ath+17], which are very high-dimensional. When finding adversarial poses, one might expect—because all renderer parameters, including the original object geometry and textures, are held constant—the success rate to depend largely on the similarities between a given 3D object and examples of the target in ImageNet. Interestingly, across our 30 objects, RS discovered 990/1000 different ImageNet classes (132 of which were shared between all objects). When only considering high-confidence ($p \geq 0.7$) misclassifications, our 30 objects were still misclassified into 797 different classes with a median number of 240 incorrect labels found per object (see Figure 5.21 and Figure 5.11 for examples). Across all adversarial poses and objects, DNNs tend to be more confident when correct than when wrong (the median of median probabilities were 0.41 vs. 0.21, respectively).

5.3.2 Common object classifications are shared across different lighting settings

Here, we analyze how our results generalize across different lighting conditions. From the data produced in Section 5.3.1 for each object, we calculated the DNN accuracy under each lighting setting. Then, for each object, we took the absolute difference of the accuracies for all three lighting combinations (i.e., bright vs. medium, bright vs. dark, and medium vs. dark) and recorded the maximum of those values. The median *maximum absolute difference* of accuracies for all objects was 2.29% (compared to the median accuracy of 3.09% across all lighting settings). That is, DNN accuracy is consistently low across all lighting conditions. Lighting changes would not alter the fact that DNNs are vulnerable to adversarial poses. We also recorded the 50 most frequent classes for each object under the different lighting settings (S_b, S_m, and S_d). Then, for each object, we computed the intersection over union



(a) (b)

Figure 5.5: Inception-v3's ability to correctly classify images is highly localized in the rotation and translation parameter space. (a) shows the classification landscape for a fire truck object when altering θ_r and θ_p and holding $(x_\delta, y_\delta, z_\delta, \theta_y)$ at $(0, 0, -3, \frac{\pi}{4})$, where light regions correspond to correct classifications while dark regions correspond to incorrect classifications. Green and red dots indicate correct and incorrect classifications, respectively, corresponding to the fire truck object poses in (b). (Image credit: [Alc+18])

score o_S for these sets:

$$o_S = 100 \times \frac{|S_b \bigcap S_m \bigcap S_d|}{|S_b \bigcup S_m \bigcup S_d|}$$

The median o_S for all objects was 47.10%. That is, for 15 out of 30 objects, 47.10% of the 50 most frequent classes were shared across lighting settings. While lighting does have an impact on DNN misclassifications (as expected), the large number of shared labels across lighting settings suggests ImageNet classes are strongly associated with certain adversarial poses regardless of lighting.

5.3.3 Correct Classifications Are Highly Localized in the Rotation and Translation Landscape

To gain some intuition for how Inception-v3 responds to rotations and translations of an object, we plotted the probability and classification landscapes for paired parameters (e.g.,

Figure 5.5, pitch vs. roll) while holding the other parameters constant. We qualitatively observed that the DNN's ability to recognize an object (e.g., a fire truck) in an image varies radically as the object is rotated in the world (Figure 5.5).

Experiment

To quantitatively evaluate the DNN's sensitivity to rotations and translations, we tested how it responded to single parameter disturbances. For each object, we randomly selected 100 distinct starting poses that the DNN had correctly classified in our random sampling runs. Then, for each parameter (e.g., yaw rotation angle), we randomly sampled 100 new values[1] while holding the others constant. For each sample, we recorded whether or not the object remained correctly classified, and then computed the failure (i.e., misclassification) rate for a given (object, parameter) pair. Plots of the failure rates for all (object, parameter) combinations can be found in Figure 5.23. Additionally, for each parameter, we calculated the median of the median failure rates. That is, for each parameter, we first calculated the median failure rate for all objects, and then calculated the median of those medians for each parameter. Further, for each (object, starting pose, parameter) triple, we recorded the magnitude of the smallest parameter change that resulted in a misclassification. Then, for each (object, parameter) pair, we recorded the median of these minimum values. Finally, we again calculated the median of these medians across objects (Table 5.1).

Results

As can be seen in Table 5.1, the DNN is highly sensitive to all single parameter disturbances, but it is especially sensitive to disturbances along the depth (z_δ), pitch (θ_p), and roll (θ_r). Note that a change in rotation as small as 8.02° can cause an object to be misclassified (refer to Table 5.1). We also observed that correctly classified poses are highly similar

[1] using the random sampling procedure described in Section 5.2.5

while misclassified poses are diverse by comparing two t-SNE plots of these two sets of poses (Figure 5.9 vs. Figure 5.11).

Parameter	Fail Rate %	Min. Δ	Int. Δ
x_δ	42	0.09	2.0
y_δ	49	0.10	4.5
z_δ	81	0.77	5.4%
θ_y	69	0.18	10.31°
θ_p	83	0.14	8.02°
θ_r	81	0.16	9.17°

Table 5.1: The median of the median failure rates and the median of the median minimum disturbances (Min. Δ) for the single parameter sensitivity tests described in Section 5.3.3. Int. Δ translates the values in Min. Δ to more interpretable units. For x_δ and y_δ, the units are pixels. For z_δ, the unit is the percent change in the area of the bounding box containing the object. See main text and Figure 5.23 for additional information.

5.3.4 Optimization methods can effectively generate targeted adversarial poses

Given a challenging, highly non-convex objective landscape (Figure 5.5), we wish to evaluate the effectiveness of two different types of approximate gradients at targeted attacks, i.e., finding adversarial examples misclassified as a target class [Sze+13]. Here, we compare (1) random search; (2) gradient descent with finite-difference gradients (FD-G); and (3) gradient descent with analytical, approximate gradients provided by a differentiable renderer (DR-G) [KUH18].

Experiment

Because our adversarial pose attacks are inherently constrained by the fixed geometry and appearances of a given 3D object (see Section 5.3.1), we defined the targets to be the 50 most frequent incorrect classes found by our RS procedure for each object. For each (object, target) pair, we ran 50 optimization trials using ZRS, FD-G, and DR-G. All treatments were initialized with a pose found by the ZRS procedure and then allowed to optimize for 100 iterations.

Results

For each of the 50 optimization trials, we recorded both whether or not the target was hit and the maximum target probability obtained during the run. For each (object, target) pair, we calculated the percent of target hits and the median maximum confidence score of the target labels (see Table 5.2). As shown in Table 5.2, FD-G is substantially more effective than ZRS at generating targeted adversarial poses, having both higher median hit rates and confidence scores. In addition, we found the approximate gradients from DR to be surprisingly noisy, and DR-G largely underperformed even non-gradient methods (ZRS) (see Section 5.2.3).

	Hit Rate %	Target Probability
ZRS random search	78	0.29
FD-G gradient-based	**92**	**0.41**
DR-G† gradient-based	32	0.22

Table 5.2: The median percent of target hits and the median of the median target probabilities for random search (ZRS), gradient descent with finite difference gradients (FD-G), and DR gradients (DR-G). All attacks are targeted and initialized with z_δ-constrained random search. †DR-G is not directly comparable to FD-G and ZRS (details in Section 5.7.3).

5.3.5 Adversarial poses transfer to different image classifiers and object detectors

The most important property of previously documented AXs is that they transfer across ML models, enabling black-box attacks [Yua+17]. Here, we investigate the transferability of our adversarial poses to (a) two different image classifiers, AlexNet [KSH12] and ResNet-50 [He+15], trained on the same ImageNet dataset; and (b) an object detector YOLOv3 [RF18] trained on the MS COCO dataset [Lin+14].

For each object, we randomly selected 1,350 AXs that were misclassified by Inception-v3 with high confidence ($p \geq 0.9$) from our untargeted RS experiments in Section 5.3.1. We exposed the AXs to AlexNet and ResNet-50 and calculated their misclassification rates. We

found that almost all AXs transfer with median misclassification rates of 99.9% and 99.4% for AlexNet and ResNet-50, respectively. In addition, 10.1% of AlexNet misclassifications and 27.7% of ResNet-50 misclassifications were identical to the Inception-v3 predicted labels. There are two orthogonal hypotheses for this result. First, the ImageNet training-set images themselves may contain a strong bias towards common poses, omitting uncommon poses (Section 5.7.6) shows supporting evidence from a nearest-neighbor test). Second, the models themselves may not be robust to even slight disturbances of the known, in-distribution poses.

Object detectors

Previous research has shown that object detectors can be more robust to adversarial attacks than image classifiers [Lu+17b]. Here, we investigate how well our AXs transfer to a state-of-the-art object detector–YOLOv3. YOLOv3 was trained on MS COCO, a dataset of bounding boxes corresponding to 80 different object classes.

We only considered the 13 objects that belong to classes present in both the ImageNet and MS COCO datasets. We found that 75.5% of adversarial poses generated for Inception-v3 are also misclassified by YOLOv3 (see Section 5.7.2 for more details). These results suggest the adversarial pose problem transfers across datasets, models, and tasks.

5.4 Adversarial training

One of the most effective methods for defending against OoD examples has been adversarial training [GSS14], i.e., augmenting the training set with AXs—also a common approach in anomaly detection [CBK09]. Here, we test whether adversarial training can improve DNN robustness to new poses generated for (1) our 30 training-set 3D objects; and (2) seven held-out 3D objects.

5.4.1 Training

We augmented the original 1,000-class ImageNet dataset with an additional 30 AX classes. Each AX class included 1,350 randomly selected high-confidence ($p \geq 0.9$) misclassified images split 1,300/50 into training/validation sets. Our AlexNet trained on the augmented dataset (AT) achieved a top-1 accuracy of 0.565 for the original ImageNet validation set and a top-1 accuracy[2] of 0.967 for the AX validation set.

	PT	AT
Error (T)	99.67	6.7
Error (H)	99.81	89.2
High-confidence Error (T)	87.8	1.9
High-confidence Error (H)	48.2	33.3

Table 5.3: The median percent of misclassifications (Error) and high-confidence (i.e., $p > 0.7$) misclassification by the pre-trained AlexNet (PT) and our AlexNet trained with adversarial examples (AT) on random poses of training-set objects (T) and held-out objects (H).

5.4.2 Evaluation

To evaluate our AT model vs. a pre-trained AlexNet (PT), we used RS to generate 10^6 samples for each of our 3D training objects. In addition, we collected seven held-out 3D objects not included in the training set that belong to the same classes as seven training-set objects (example renders in Figure 5.19). We followed the same sampling procedure for the held-out objects to evaluate whether our AT generalizes to unseen objects.

For each of these $30 + 7 = 37$ objects and for both the PT and our AT, we recorded two statistics: (1) the percent of misclassifications, i.e., errors; and (2) the percent of high-confidence (i.e., $p \geq 0.7$) misclassification (Table 5.3). Following adversarial training, the accuracy of the DNN substantially increased for *known* objects (Table 5.3; 99.67% vs. 6.7%).

[2]In this case, a classification was *correct* if it matched *either* the original ImageNet positive label *or* the negative, object label.

However, our AT still misclassified the adversarial poses of held-out objects at an 89.2% error rate. We hypothesize that augmenting the dataset with many more 3D objects may improve DNN generalization on held-out objects. Here, AT might have used (1) the grey background to separate the 1,000 original ImageNet classes from the 30 AX classes; and (2) some non-geometric features sufficient to discriminate among only 30 objects. However, as suggested by our work in Section 5.2.4, acquiring a large-scale, high-quality 3D object dataset is costly and labor-intensive. Currently, no such public dataset exists, and thus we could not test this hypothesis.

5.5 Related work

5.5.1 Out-of-distribution Detection

OoD classes, i.e., classes not found in the training set, present a significant challenge for computer vision technologies in real-world settings [Sch+13]. Here, we study an orthogonal problem—correctly classifying OoD poses of objects from *known* classes. While rejecting to classify is a common approach for handling OoD examples [HG17; Sch+13], the OoD poses in our work come from known classes and thus *should be* assigned correct labels.

5.5.2 2D Adversarial Examples

Numerous techniques for crafting AXs that fool image classifiers have been discovered [Yua+17]. However, previous work has typically optimized in the 2D input space [Yua+17], e.g., by synthesizing an entire image [NYC14], a small patch [KZG18; Evt+17], a few pixels [CW16], or only a single pixel [SVK17]. But pixel-wise changes are uncorrelated [Ngu+17], so pixel-based attacks may not transfer well to the real world [Lu+17a; Luo+15] because there is an infinitesimal chance that such specifically crafted, uncorrelated pixels will be encountered in the vast physical space of camera, lighting, traffic, and weather configurations.

5.5.3 3D Adversarial Examples

[Ath+17] used a 3D renderer to synthesize textures for a 3D object such that, under a wide range of camera views, the object was still rendered into an effective AX. We also used 3D renderers, but instead of optimizing textures, we optimized the poses of known objects to cause DNNs to misclassify (i.e., we kept the textures, lighting, camera settings, and background image constant).

5.5.4 Concurrent work

We describe below two concurrent attempts that are closely related but orthogonal to our work. First, [Liu+18] proposed a differentiable 3D renderer and used it to perturb both an object's geometry and the scene's lighting to cause a DNN to misbehave. However, their geometry perturbations were constrained to be infinitesimal so that the visibility of the vertices would not change. Therefore, their result of minutely perturbing the geometry is effectively similar to that of perturbing textures [Ath+17]. In contrast, we performed 3D rotations and 3D translations to move an object inside a 3D space (i.e., the viewing frustum of the camera).

Second, an anonymous ICLR 2019 submission [Ano19] showed how simple rotations and translations of an image can cause DNNs to misclassify. However, these manipulations were still applied to the entire 2D image and thus do not reveal the type of adversarial poses discovered by rotating 3D objects (e.g., a flipped-over school bus; Figure 5.1 (d)).

To the best of our knowledge, our work is the first attempt to harness 3D objects to study the OoD poses of well-known training-set objects that cause state-of-the-art ImageNet classifiers and MS COCO detectors to misclassify.

5.6 Discussion and Conclusion

In this paper, we revealed how DNNs' understanding of objects like `school bus` and `fire truck` is quite naive—they can correctly label only a small subset of the entire pose space for 3D objects. Note that we can also find real-world OoD poses by simply taking photos of real objects (Figure 5.22).

We believe classifying an arbitrary pose into one of the object classes is an ill-posed task, and that the adversarial pose problem might be alleviated via multiple orthogonal approaches. The first is addressing biased data [TE11]. Because ImageNet and MS COCO datasets are constructed from photographs taken by people, the datasets reflect the aesthetic tendencies of their captors. Such biases can be somewhat alleviated through data augmentation, specifically, by harnessing images generated from 3D renderers [Shr+16; Alh+18].

From the modeling view, we believe DNNs would also benefit from strong 3D geometric priors [Alh+18]. Finally, our work introduced a new promising method (Figure 5.2) for testing computer vision DNNs by harnessing 3D renderers and 3D models. While we only optimize a single object here, the framework could be extended to jointly optimize lighting, background image, and multiple objects, all in one *adversarial world*. Not only does our framework enable us to enumerate test cases for DNNs, but it also serves as an interpretability tool for extracting useful insights about these black-box models' inner functions.

5.7 Appendix

5.7.1 Extended Description of the 3D Object Dataset and Its Evaluation

Dataset Construction

1. Classes Our main dataset consists of 30 unique 3D object models corresponding to 30 ImageNet classes relevant to a traffic environment. The 30 classes include 20 vehicles (e.g., `school bus` and `cab`) and 10 street-related items (e.g., `traffic light`). See Figure 5.7 for example renders of each object.

2. Acquisition We collected 3D objects and constructed our own datasets for the study. 3D models with high-quality image textures were purchased from `https://www.turbosquid.com`, `https://free3d.com`, and `https://www.cgtrader.com`. To make sure the renders were as close to real ImageNet photos as possible, we used only 3D models that had high-quality 2D image textures. We did not choose 3D models from public datasets, e.g., ObjectNet3D [Xia+16], because most of them do not have high-quality image textures. While the renders of such models may be correctly classified by DNNs, we excluded them from our study because of their poor realism. We also examined the ImageNet images to ensure they contained real-world examples qualitatively similar to each 3D object in our 3D dataset.

3. 3D Objects Each 3D object is represented as a mesh, i.e., a list of triangular faces, each defined by three vertices [MS15]. The 30 meshes have on average 9,908 triangles (see Table 5.4 for specific numbers).

Manual Object Tessellation for Experiments Using the Differentiable Renderer

In contrast to ModernGL [Dom19] (the non-differentiable renderer (NR) in our paper), the differentiable renderer (DR) [KUH18] does not perform tessellation, a standard process to increase the resolution of renders. Therefore, the render quality of the DR is lower than

3D object	N_T	N_O	3D object	N_T	N_O
ambulance	70,228	5,348	motor scooter	96,638	2,356
backpack	48,251	1,689	moving van	83,712	5,055
bald eagle	63,212	2,950	park bench	134,162	1,972
beach wagon	220,956	2,024	parking meter	37,246	1,086
cab	53,776	4,743	pickup	191,580	2,058
cellphone	59,910	502	police van	243,132	1,984
fire engine	93,105	8,996	recreational vehicle	191,532	1,870
forklift	130,455	5,223	school bus	229,584	6,244
garbage truck	97,482	5,778	sports car	194,406	2,406
German shepherd	88,496	88,496	street sign	17,458	17,458
golf cart	98,007	5,153	tiger cat	107,431	3,954
jean	17,920	17,920	tow truck	221,272	5,764
jeep	191,144	2,282	traffic light	392,001	13,840
minibus	193,772	1,910	trailer truck	526,002	5,224
minivan	271,178	1,548	umbrella	71,410	71,410

Table 5.4: The triangle number for the 30 objects used in our study. N_O shows the number of triangles for the original 3D objects, and N_T shows the same number after tessellation. Across 30 objects, the average triangle count increases $\sim$ 15x from $\overline{N_O} = 9,908$ to $\overline{N_T} = 147,849$.

that of the NR. To minimize this gap and make results from the NR more comparable with those from the DR, we manually tessellated each 3D object as a pre-processing step for rendering with the DR. Using the manually tessellated objects, we then (1) evaluated the render quality of the DR (Section 5.7.1); and (2) performed research experiments with the DR (i.e., the DR-G method in Section 5.3.4).

1. Tessellation We used the *Quadify Mesh Modifier* feature (quad size of 2%) in 3ds˜Max 2018 to tessellate objects, increasing the average number of faces $\sim$15x from $9,908$ to $147,849$ (see Table 5.4). The render quality after tessellation is sharper and of a higher resolution (see Figure 5.6 (a) vs. (b)). Note that the NR pipeline already performs tessellation for every input 3D object. Therefore, we did not perform manual tessellation for 3D objects rendered by the NR.

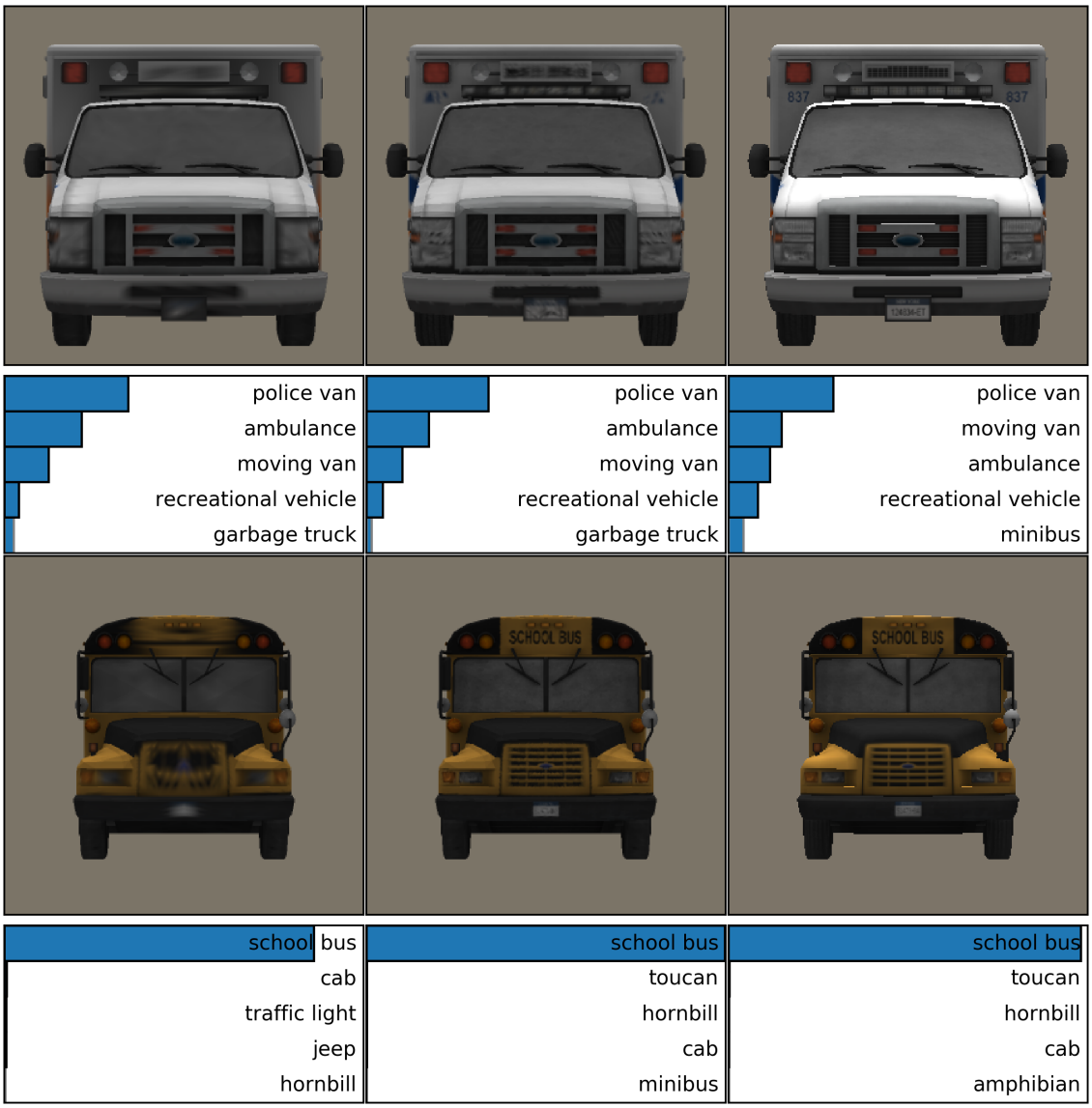


Figure 5.6: A comparison of 3D object renders (here, `ambulance` and `school bus`) before and after tessellation. (a) Original 3D models rendered by the differentiable renderer (DR) [KUH18] without tessellation. (b) DR renderings of the same objects after manual tessellation. (c) The non-differentiable renderer (NR), i.e., ModernGL [Dom19], renderings of the original objects. After manual tessellation, the render quality of the DR appears to be sharper ((a) vs. (b)) and closely matches that of the NR, which also internally tessellates objects ((b) vs. (c)). (Image credit: [Alc+18])

Evaluation

We recognize that a reality gap will often exist between a render and a real photo.

Therefore, we rigorously evaluated our renders to make sure the reality gap was acceptable for our study. From ∼100 initially-purchased 3D object models, we selected the 30 highest-quality objects that both (1) passed a visual human Turing test; and (2) were correctly recognized with high confidence by the Inception-v3 classifier [Sze+16].

1. Qualitative Evaluation We did not use the 30 objects chosen for the main dataset (Section 5.7.1) to evaluate the *general* quality of the DR renderings of high-quality objects on realistic background images. Instead, we randomly chose a separate set of 17 high-quality image-textured objects for evaluation. Using the 17 objects, we generated 56 renders that matched 56 reference (real) photos. Then, we qualitatively evaluated the renders both separately and in a side-by-side comparison with real photos. Specifically, we produced 56 (real photo, render) pairs (see Figure 5.8) via the following steps:

   (a) We retrieved ∼3 real photos for each 3D object (e.g., a car) from the Internet (using descriptive information, e.g., a car's make, model, and year).

   (b) For each real photo, we replaced the object with matching background content via Adobe Photoshop's Context-Aware Fill-In feature to obtain a background-only photo B (i.e., no foreground objects).

   (c) We rendered the 3D object with the differentiable renderer on the background B obtained in Step 2. We then manually aligned the pose of the 3D object such that it closely matched that in the reference photo.

   (d) We evaluated pairs of (photo, render) in a side-by-side comparison.

   While discrepancies can be visually spotted in our side-by-side comparisons, we found that most of the renders passed our human visual Turing test if presented alone. That is, it is not easy for humans to tell whether a render is a real photo or not (if they

are not primed with the reference photos). We only show pairs rendered by the DR because the NR qualitatively has a slightly higher rendering quality (Figure 5.6 (b) vs. (c)).

2. Quantitative Evaluation In addition to the qualitative evaluation, we also quantitatively evaluated the Google Inception-v3 [Sze+16] top-1 accuracy on renders that use either an empty background or real background images.

3. Evaluation of the Renders of 30 Objects on An Empty Background Because the experiments in the main text used our self-assembled 30-object dataset (Section 5.7.1), we describe the process and the results of our quantitative evaluation for only those objects.

   We rendered the objects on a white background with RGB values of (1.0, 1.0, 1.0), an ambient light intensity of 0.9, and a directional light intensity of 0.5. For each object, we sampled 36 unique views (common in ImageNet) evenly divided into three sets. For each set, we set the object at the origin, the up direction to $(0, 1, 0)$, and the camera position to $(0, 0, -z)$ where $z = \{4, 6, 8\}$. We sampled 12 views per set by starting the object at a 10° yaw and generating a render at every 30° yaw-rotation. Across all objects and all renders, the Inception-v3 top-1 accuracy is 83.23% (comparable to 77.45% on ImageNet images [Sze+16]) with a mean top-1 confidence score of 0.78. The top-1 and top-5 average accuracy and confidence scores are shown in Table 5.5.

Distance	4	6	8	Average
top-1 mean accuracy	84.2%	84.4%	81.1%	83.2%
top-5 mean accuracy	95.3%	98.6%	96.7%	96.9%
top-1 mean confidence score	0.77	0.80	0.76	0.78

Table 5.5: The top-1 and top-5 average accuracy and confidence scores for Inception-v3 [Sze+16] on the renders of the 30 objects in our dataset.

4. Evaluation of the Renders of Test Objects on Real Backgrounds In addition to our qualitative side-by-side (real photo, render) comparisons (Figure 5.8), we quantitatively compared Inception-v3's predictions for our renders to those for real photos.

   We found a high similarity between real photos and renders for DNN predictions. That is, across all 56 pairs (Section 1), the top-1 predictions match 71.43% of the time. Across all pairs, 76.06% of the top-5 labels for real photos match those for renders.

5.7.2 Transferability from the Inception-v3 Classifier to the YOLO-v3 Detector

Previous research has shown that object detectors can be more robust to adversarial attacks than image classifiers [Lu+17b]. Here, we investigate how well our AXs generated for an Inception-v3 classifier trained to perform 1,000-way image classification on ImageNet [Rus+15] transfer to YOLO-v3, a state-of-the-art object detector trained on MS COCO [Lin+14].

Note that while ImageNet has 1,000 classes, MS COCO has bounding boxes classified into only 80 classes. Therefore, among 30 objects, we only selected the 13 objects that (1) belong to classes found in both the ImageNet and MS COCO datasets; and (2) are also well recognized by the YOLO-v3 detector in common poses.

Class mappings from ImageNet to MS COCO

See Table 5.6 (a) for 13 mappings from ImageNet labels to MS COCO labels.

Selecting 13 Objects for the Transferability Test

For the transferability test (Section 5.7.2), we identified the 13 objects (out of 30) that are well detected by the YOLO-v3 detector via the two tests described below.

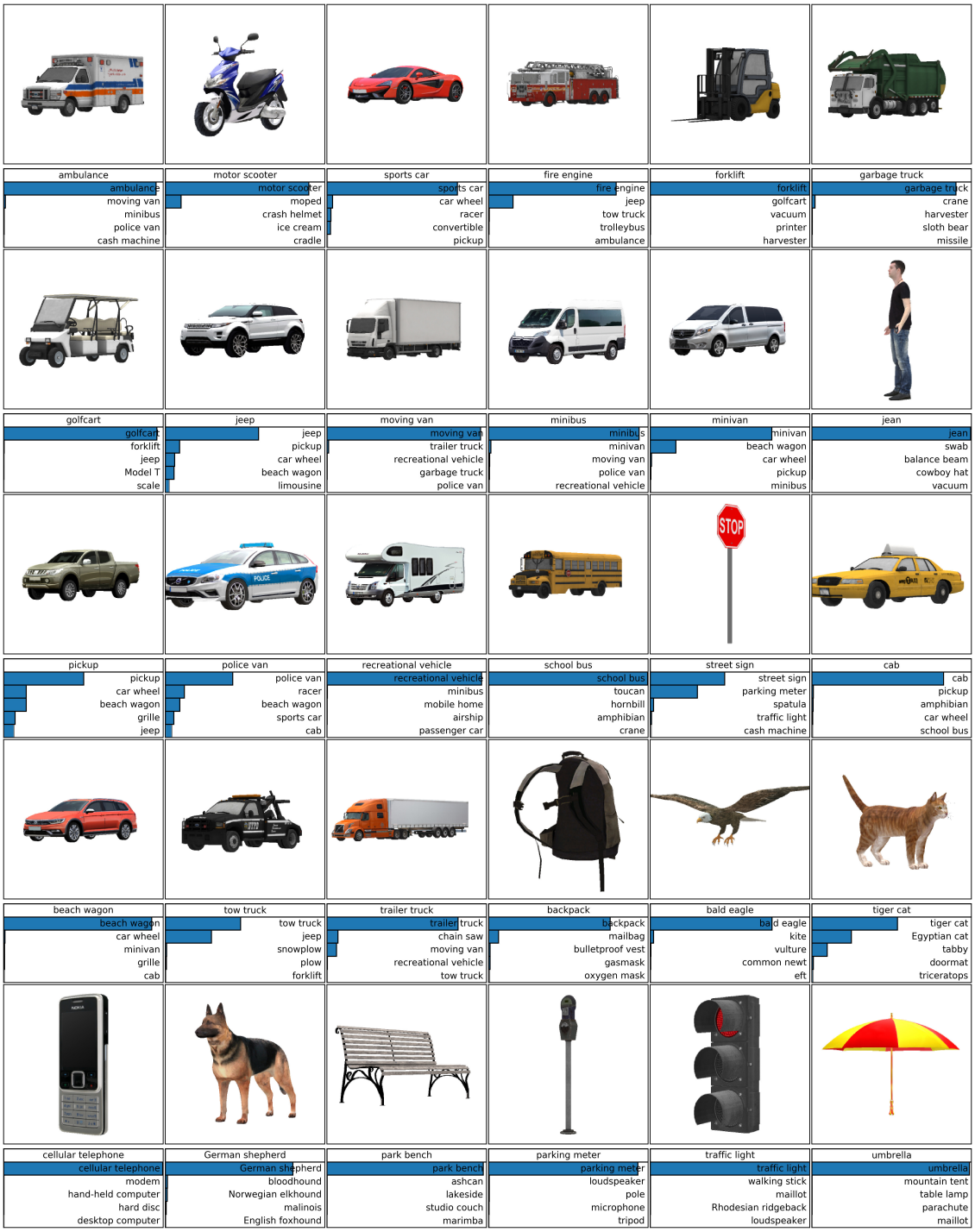


Figure 5.7: We tested Inception-v3's predictions on the renders generated by the differentiable renderer (DR). We show here the top-5 predictions for one random pose per object. However, in total, we generated 36 poses for each object by (1) varying the object distance to the camera; and (2) rotating the object around the yaw axis. See `https://goo.gl/7LG3Cy` for all the renders and DNN top-5 predictions. Across all 30 objects, on average, Inception-v3 correctly recognizes 83.2% of the renders. See Section 2 for more details. (Image credit: [Alc+18])

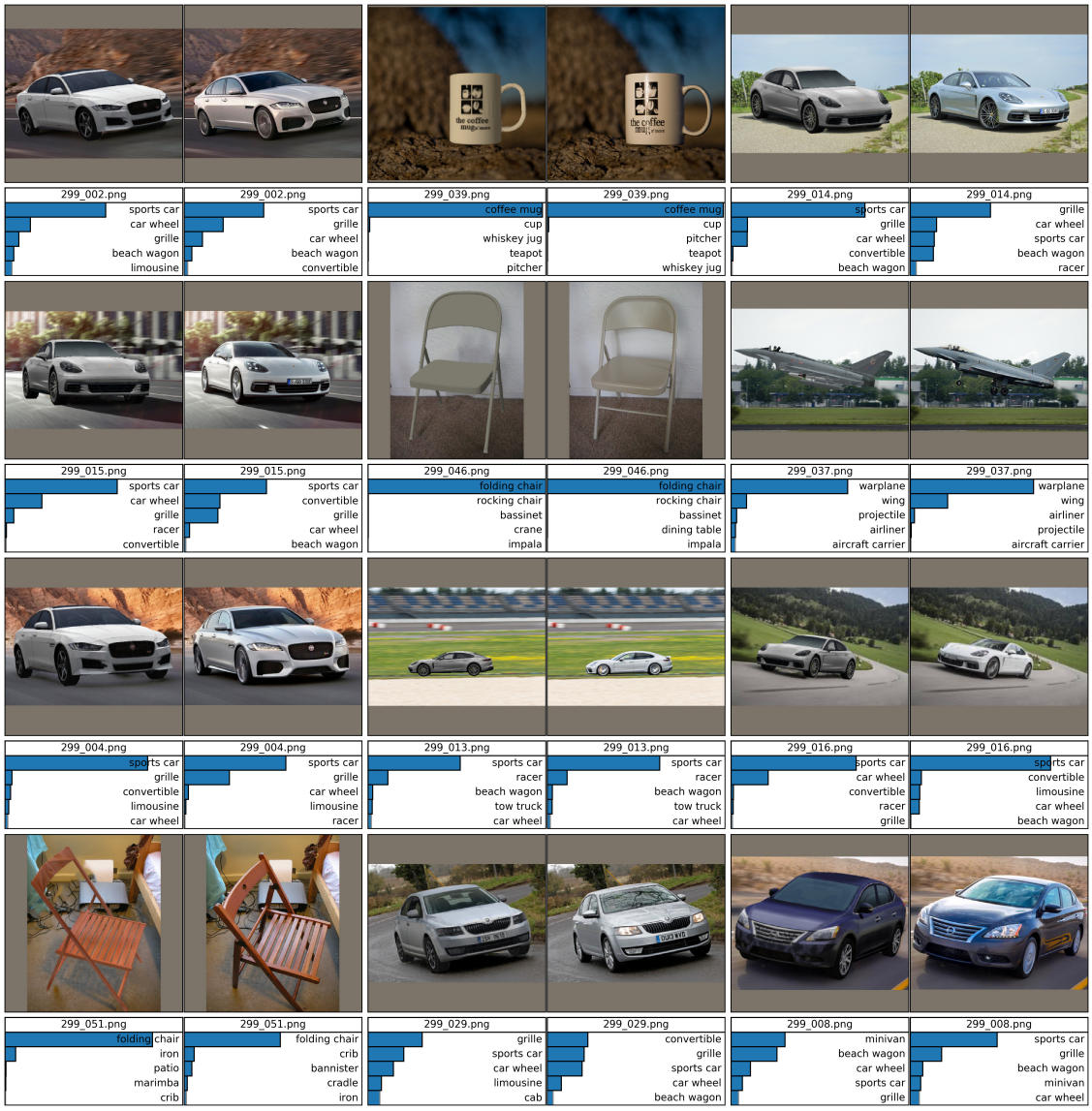


Figure 5.8: 12 random pairs of renders (left) and real photos (right) among 56 pairs produced in total for our 3D object rendering evaluation (Section 1). The renders are produced by the differentiable renderer by [KUH18]. More images are available at `https://goo.gl/8z42zL`. While discrepancies can be spotted in our side-by-side comparisons, we found that most of the renders passed our human visual Turing test if presented alone. (Image credit: [Alc+18])

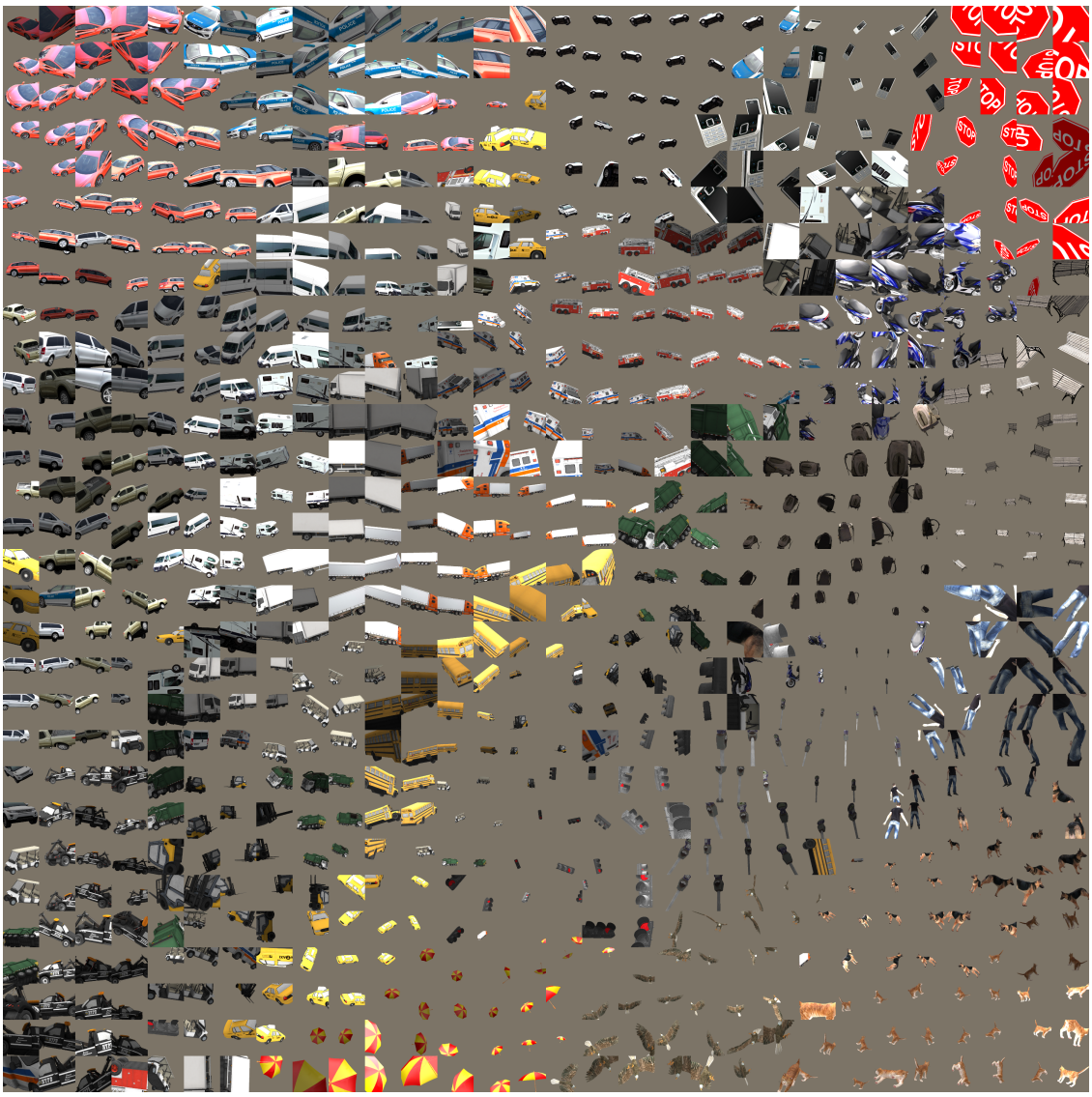


Figure 5.9: For each object, we collected 30 high-confidence ($p \geq 0.9$) correctly classified images by Inception-v3. The images were generated via the random search procedure. We show here a grid t-SNE of AlexNet [KSH12] fc7 features for all 30 objects $\times$ 30 images = 900 images. Correctly classified images for each object tend to be similar and clustered together. The original, high-resolution figure is available at `https://goo.gl/TGgPgB`. To better visualize the clusters, we plotted the same t-SNE but used unique colors to denote the different 3D objects in the renders (Figure 5.10). Compare and contrast this plot with the t-SNE of only misclassified poses (Figure 5.11 and Figure 5.12). (Image credit: [Alc+18])

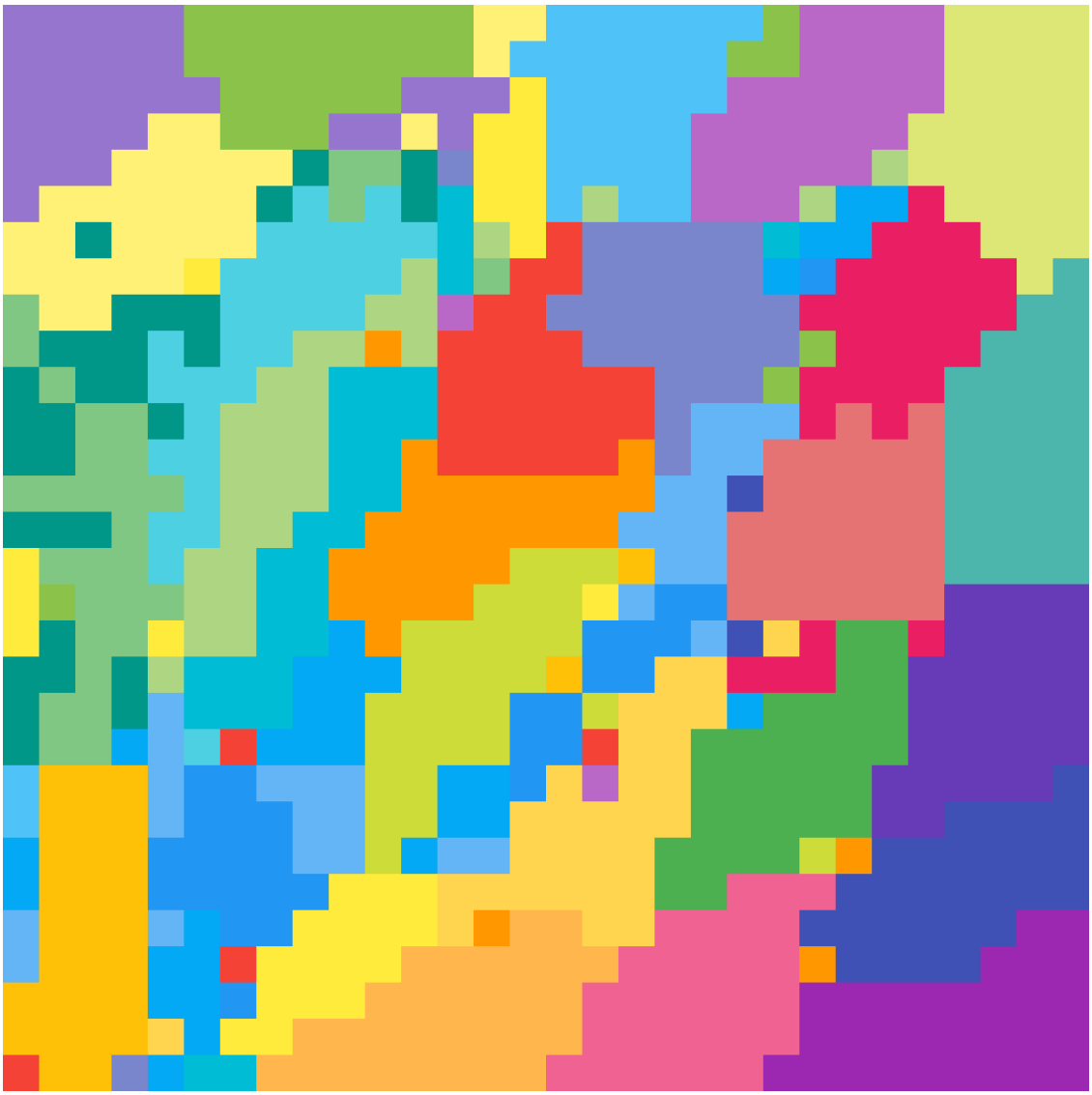


Figure 5.10: The same t-SNE found in Figure 5.9 but using a unique color to denote the 3D object found in each rendered image. Here, each color also corresponds to a unique Inception-v3 label. Compare and contrast this plot with the t-SNE of only misclassified poses (Figure 5.12). The original high-resolution figure is available at `https://goo.gl/TGgPgB`. (Image credit: [Alc+18])

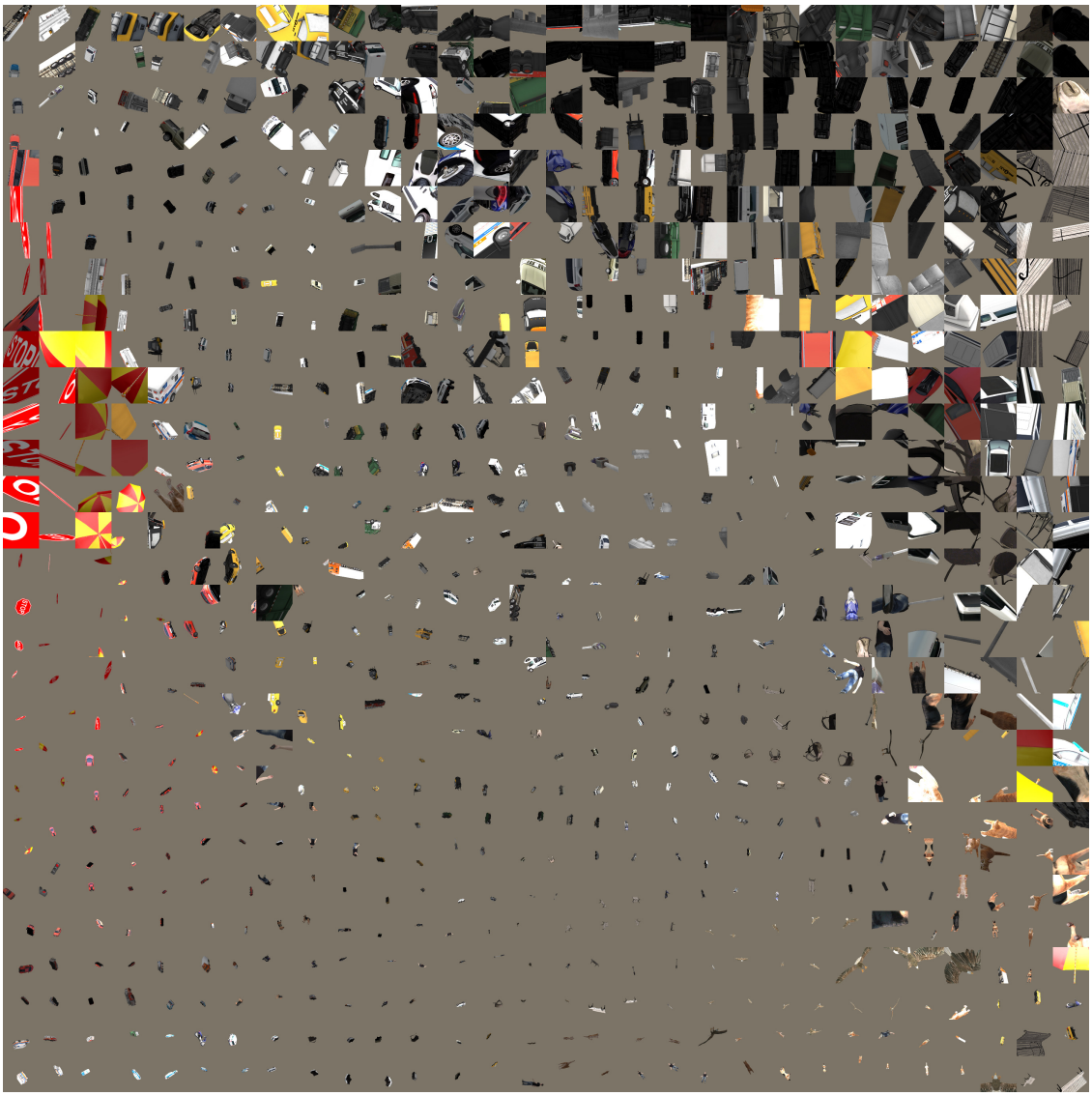


Figure 5.11: Following the same process as described in Figure 5.9, we show here a grid t-SNE of generated adversarial poses. For each object, we assembled 30 high-confidence ($p \geq 0.9$) adversarial examples generated via a random search against Inception-v3 [Sze+16]. The t-SNE was generated from the AlexNet [KSH12] `fc7` features for 30 objects $\times$ 30 images $=$ 900 images. The original, high-resolution figure is available at `https://goo.gl/TGgPgB`. Adversarial poses were found to be both common across different objects (e.g., the top-right corner) and unique to specific objects (e.g., the `traffic sign` and `umbrella` objects in the middle left). To better understand how similar misclassified poses can be found across many objects, see Figure 5.12. Compare and contrast this plot with the t-SNE of correctly classified poses (Figure 5.9 and Figure 5.10). (Image credit: [Alc+18])

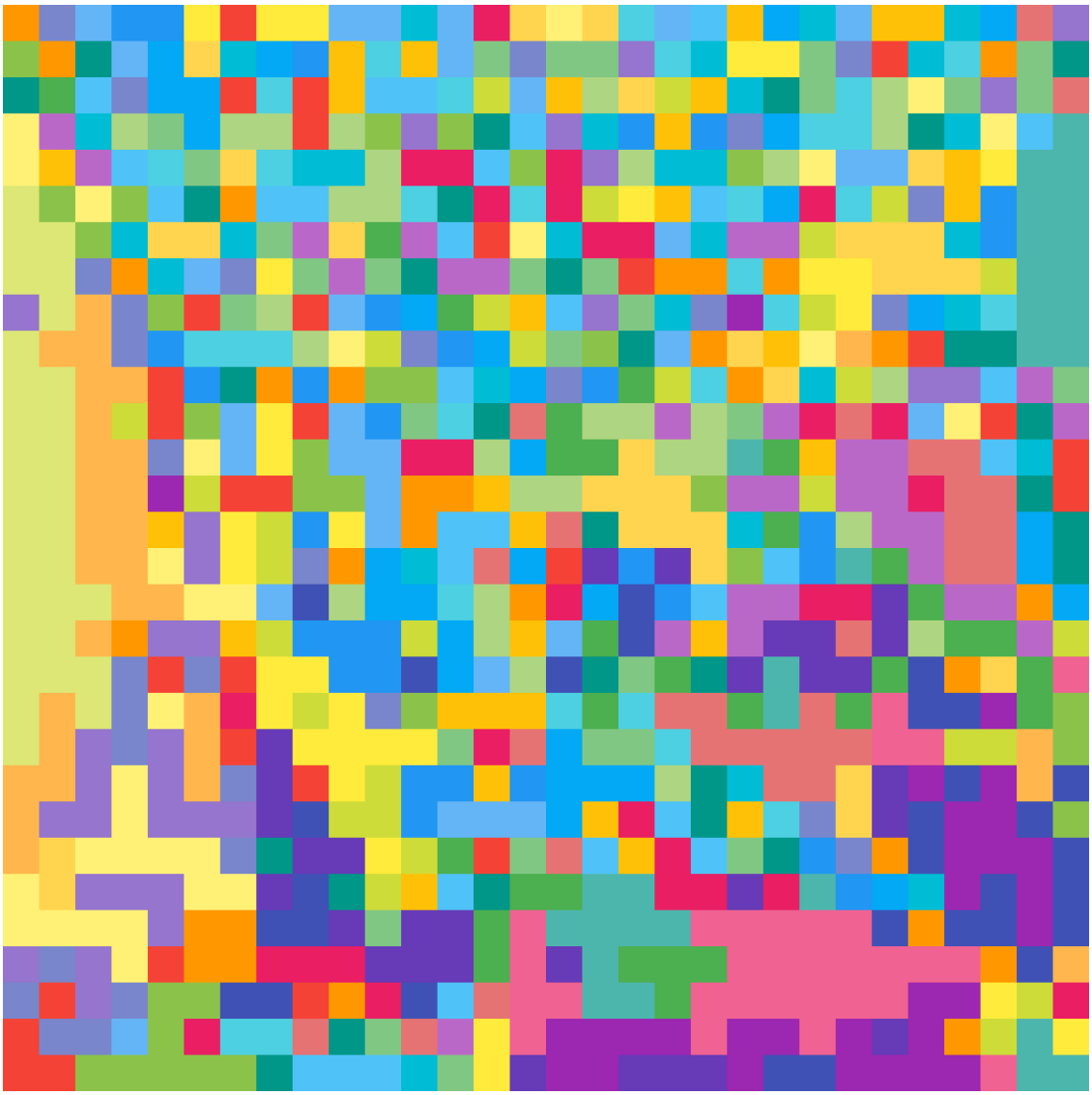


Figure 5.12: The same t-SNE as that in Figure 5.11 but using a unique color to denote the 3D object used to render the adversarial image (i.e., Inception-v3's misclassification labels are not shown here). The original, high-resolution figure is available at `https://goo.gl/TGgPgB`. Compare and contrast this plot with the t-SNE of correctly classified poses (Figure 5.10). (Image credit: [Alc+18])

YOLO-v3 Correctly Classifies 93.80% of Poses Generated via Yaw-rotation

We rendered 36 unique views for each object by generating a render at every 30° yaw-rotation (see Section 2). Note that, across all objects, these yaw-rotation views have an average accuracy of 83.2% by the Inception-v3 classifier. We tested them against YOLO-v3 to see whether the detector was able to correctly find one single object per image and label it correctly. Among 30 objects, we removed those that YOLO-v3 had an accuracy $\leq$70%, leaving 13 for the transferability test. Across the remaining 13 objects, YOLO-v3 has an accuracy of 93.80% on average (with an NMS threshold of 0.4 and a confidence threshold of 0.5). Note that the accuracy was computed as the total number of correct labels over the total number of bounding boxes detected (i.e., we did not measure bounding-box IoU errors). See class-specific statistics in Table 5.6. This result shows that YOLO-v3 is substantially more accurate than Inception-v3 on the standard object poses generated by yaw-rotation (93.80% vs. 83.2%).

YOLO-v3 Correctly Classifies 81.03% of Poses Correctly Classified by Inception-v3

Additionally, as a sanity check, we tested whether poses *correctly classified* by Inception-v3 transfer well to YOLO-v3. For each object, we randomly selected 30 poses that were 100% correctly classified by Inception-v3 with high confidence ($p \geq 0.9$). The images were generated via the random search procedure in the main text experiment (Section 5.2.5). Across the final 13 objects, YOLO-v3 was able to correctly detect one single object per image and label it correctly at a 81.03% accuracy (see Table 5.6 (c)).

Transferability Test: YOLO-v3 Fails on 75.5% of Adversarial Poses Misclassified by Inception-v3

For each object, we collected 1,350 random adversarial poses (i.e., incorrectly classified by Inception-v3) generated via the random search procedure (Section 5.2.5). Across all 13 objects and all adversarial poses, YOLO-v3 obtained an accuracy of only 24.50% (compared to 81.03% when tested on images correctly classified by Inception-v3). In other words, 75.5% of adversarial poses generated for Inception-v3 also escaped the detection[3] of YOLO-v3 (see Table 5.6 (d) for class-specific statistics). Our result shows adversarial poses transfer well across tasks (image classification $\rightarrow$ object detection), models (Inception-v3 $\rightarrow$ YOLO-v3), and datasets (ImageNet $\rightarrow$ MS COCO).

5.7.3 Experimental Setup for the Differentiable Renderer

For the gradient descent method (DR-G) that uses the approximate gradients provided by the differentiable renderer (DR) [KUH18], we set up the rendering parameters in the DR to closely match those in the NR. However, there were still subtle discrepancies between the DR and the NR that made the results (DR-G vs. FD-G in Section 5.3.4) not directly comparable. Despite these discrepancies (described below), we still believe the FD gradients are more stable and informative than the DR gradients (i.e., FD-G outperformed DR-G)[4].

DR Setup

For all experiments with the DR, the camera was centered at $(0, 0, 16)$ with an up direction $(0, 1, 0)$. The object's spatial location was constrained such that the object center was always within the frame. The depth values were constrained to be within $[-14, 14]$. Similar to experiments with the NR, we used the medium lighting setting.

[3]We were not able to check how many misclassification labels by YOLO-v3 were the same as those by Inception-v3 because only a small set of 80 the MS COCO classes overlap with the 1,000 ImageNet classes.

[4]In preliminary experiments with only the DR (not the NR), we also empirically found FD-G to be more stable and effective than DR-G (data not shown).

	(a) Label mapping		(b) Yaw Acc		(c) Rand Acc		(d) Adv Acc		
	ImageNet	MS COCO	#/36	acc	#/30	acc	#/1350	acc	Δacc
1	park bench	bench	31	86.11	22	73.33	211	15.63	57.70
2	bald eagle	bird	34	94.11	24	80.00	597	44.22	35.78
3	school bus	bus	36	100.00	18	60.00	4	0.30	69.70
4	beach wagon	car	34	94.44	30	100.00	232	17.19	82.81
5	tiger cat	cat	26	72.22	25	83.33	181	13.41	69.93
6	German shepherd	dog	32	88.89	28	93.33	406	30.07	63.26
7	motor scooter	motorcycle	36	100.00	18	60.00	384	28.44	31.56
8	jean	person	36	100.00	29	96.67	943	69.85	26.81
9	street sign	stop sign	31	86.11	26	86.67	338	25.04	61.15
10	moving van	truck	36	100.00	24	80.00	15	1.11	78.89
11	umbrella	umbrella	35	97.22	25	83.33	907	67.19	16.15
12	police van	car	36	100.00	25	83.33	55	4.07	79.26
13	trailer truck	truck	36	100.00	22	73.33	26	1.93	71.41
		Average		93.80		81.03		24.50	56.53

Table 5.6: Adversarial poses generated for a state-of-the-art ImageNet image classifier (here, Inception-v3) transfer well to an MS COCO detector (here, YOLO-v3). The table shows the YOLO-v3 detector's accuracy on: (b) object poses generated by a standard process of yaw-rotating the object; (c) random poses that are 100% correctly classified by Inception-v3 with high confidence ($p \geq 0.9$); and (d) adversarial poses, i.e., 100% misclassified by Inception-v3.

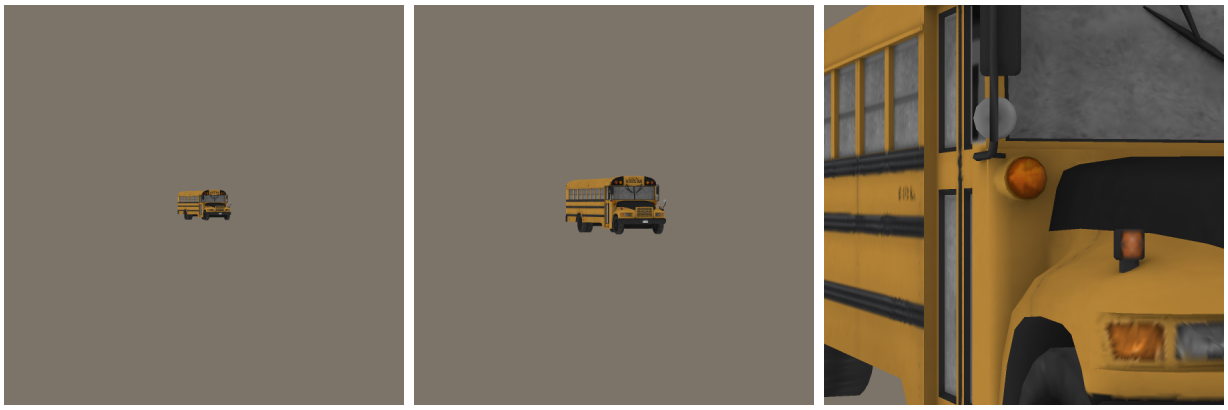
1. The mappings of 13 ImageNet classes onto 12 MS COCO classes.

2. The accuracy ("acc (%)") of the YOLO-v3 detector on 36 yaw-rotation poses per object.

3. The accuracy of YOLO-v3 on 30 random poses per object that were correctly classified by Inception-v3.

4. The accuracy of YOLO-v3 on 1,350 adversarial poses ("acc (%)") and the differences between c and d ("Δacc (%)").

The ambient light color was set to white with an intensity 1.0, while the directional light was set to white with an intensity 0.4. Figure 5.13 shows an example school bus rendered under this medium lighting at different distances.



(a) School bus at $(0, 0, -14)$ (b) School bus at $(0, 0, 0)$ (c) School bus at $(0, 0, 14)$

Figure 5.13: School bus rendered by the DR at different distances. (Image credit: [Alc+18])

The known discrepancies between the experimental setups of FD-G (with the NR) vs. DR-G (with the DR) are:

1. The exact medium lighting parameters for the NR described in the main text (Section 5.3.1) did not produce similar lighting effects in the DR. Therefore, the DR lighting parameters described above were the result of manually tuning to qualitatively match the effect produced by the NR medium lighting parameters.

2. While the NR uses a built-in tessellation procedure that automatically tessellates input objects before rendering, we had to perform an extra pre-processing step of manually tessellating each object for the DR. While small, a discrepancy still exists between the two rendering results (Figure 5.6 (b) vs. (c)).

5.7.4 Gradient Descent with the DR Gradients

In preliminary experiments (data not shown), we found the DR gradients to be relatively noisy when using gradient descent to find targeted adversarial poses (i.e., DR-G experiments). To mitigate this problem, we experimented with

1. parameter augmentation, Section 5.7.4;

2. multi-view optimization, Section 5.7.4.

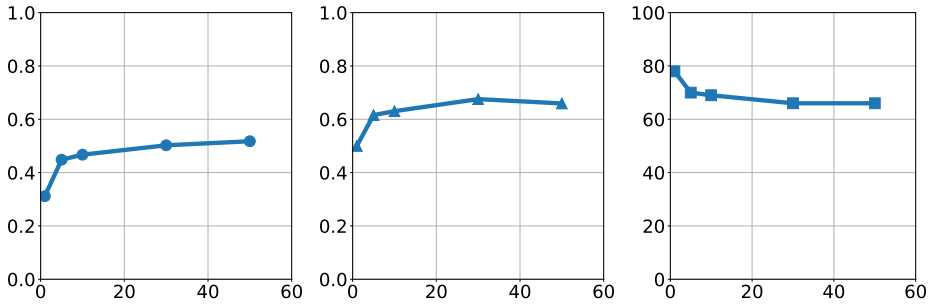
In short, we found parameter augmentation helped and used it in DR-G. However, when using the DR, we did not find multiple cameras improved optimization performance and thus only performed regular single-view optimization for DR-G.

Parameter augmentation

We performed gradient descent using the DR gradients (DR-G) in an augmented parameter space corresponding to 50 rotations and one translation to be applied to the original object vertices. That is, we backpropagated the DR gradients into the parameters of these pre-defined transformation matrices. Note that DR-G is given the same budget of 100 steps per optimization run as FD-G and ZRS for comparison in Section 5.3.4. The final transformation matrix is constructed by a series of rotations followed by one translation.

$$M = T \cdot R_{n-1} R_{n-2} \cdots R_0$$

Where M is the final transformation matrix, R_i the rotation matrices, and T the translation matrix. We empirically found that increasing the number of rotations per step helped (a) improve the success rate of hitting the target labels; (b) increase the maximum confidence score of the found AXs; and (c) reduce the number of steps, i.e., led to faster convergence (see Figure 5.14). Therefore, we empirically chose $n = 50$ for all DR-G experiments reported in the main text.



(a) y-axis: success rate (b) y-axis: max confidence (c) y-axis: mean number of steps

Figure 5.14: We found that increasing the number of rotations (displayed in x-axes) per step helped (Image credit: [Alc+18]):

1. improve the success rate of hitting the target labels;
2. increase the maximum confidence score of the found adversarial examples;
3. reduce the average number of steps required to find an AX, i.e., led to faster convergence.

Multi-view optimization

Additionally, we attempted to harness multiple views (from multiple cameras) to increase the chance of finding a target adversarial pose. Multi-view optimization did not outperform single-view optimization using the DR in our experiments. Therefore, we only performed regular single-view optimization for DR-G. We briefly document our negative results below.

Instead of backpropagating the DR gradient to a single camera looking at the object in the 3D scene, one may set up multiple cameras, each looking at the object from a different angle. This strategy intuitively allows gradients to still be backpropagated into the vertices that may be occluded in one view but visible in some other view. We experimented with six cameras and backpropagating to all cameras in each step. However, we only updated the object following the gradient from the view that yielded the lowest loss among all views. One hypothesis is that having multiple cameras might improve the chance of hitting the target. In our experiments with the DR using 100 steps per optimization run, multi-view optimization

performed worse than single-view in terms of both the success rate and the number of steps to converge. We did not compare all 30 objects due to the expensive computational cost, and only report the results from optimizing two objects `bald eagle` and `tiger cat` in Table 5.7. Intuitively, multi-view optimization might outperform single-view optimization given a large enough number of steps.

	`bald eagle`		`tiger cat`	
	Steps	Success rate	Steps	Success rate
Single-view	71.80	0.44	90.70	0.15
Multi-view	81.28	0.23	96.84	0.04

Table 5.7: Multi-view optimization performed worse than single-view optimization in both (a) the number of steps to converge and (b) success rates. We show here the results of two runs of optimizing with the `bald eagle` and `tiger cat` objects. The results are averaged over 50 target labels $\times$ 50 trials = 2,500 trials. Each optimization trial for both single- and multi-view settings is given the budget of 100 steps.

5.7.5 3D Transformation Matrix

A rotation of θ around an arbitrary axis (x, y, z) is given by the following homogeneous transformation matrix.

$$R = \begin{vmatrix} xx(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ xy(1-c)+zs & yy(1-c)+c & yz(1-c)-xs & 0 \\ xz(1-c)-ys & yz(1-c)+xs & yz(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{5.5}$$

Where $s = \sin\theta$, $c = \cos\theta$, and the axis is normalized, i.e., $x^2 + y^2 + z^2 = 1$. Translation by a vector (x, y, z) is given by the following homogeneous transformation matrix.

$$T = \begin{vmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{5.6}$$

Note that in the optimization experiments with random search (RS) and finite-difference gradients (FD-G), we dropped the homogeneous component for simplicity, i.e., the rotation matrices of yaw, pitch, and roll are all 3 by 3. The homogeneous component is only necessary for translation, which can be achieved via simple vector addition. However, in DR-G, we used the homogeneous component because we had some experiments interweaving translation and rotation. The matrix representation was more convenient for the DR-G experiments. As they are mathematically equivalent, this arbitrary implementation choice should not alter our results.

Object	Acc (%)	Object	Acc (%)	Object	Acc (%)
ambulance	3.64	golfcart	2.14	police van	0.95
backpack	8.63	jean	2.71	R.V.	2.05
bald eagle	13.26	jeep	0.29	school bus	3.48
beach wagon	0.60	minibus	0.83	sports car	2.50
cab	2.64	minivan	0.66	street sign	26.32
cell phone	14.97	motor scooter	20.49	tiger cat	7.36
fire engine	4.31	moving van	0.45	tow truck	0.87
forklift	5.20	park bench	5.72	traffic light	14.95
garbage truck	4.88	parking meter	1.27	trailer truck	1.27
German shepherd	9.61	pickup	0.86	umbrella	49.88

Table 5.8: The percent of three million random samples that were correctly classified by Inception-v3 [Sze+16] for each object. That is, for each lighting setting in $\{\mathsf{bright}, \mathsf{medium}, \mathsf{dark}\}$, we generated 10^6 samples. See Section 5.2.5 for details on the sampling procedure.

(a) bright (b) medium (c) dark

Figure 5.15: Renders of the `school bus` object using the NR [Dom19] at three different lighting settings. The directional light intensities and ambient light intensities were $(1.2, 1.6)$, $(0.4, 1.0)$, and $(0.2, 0.5)$ for the bright, medium, and dark settings, respectively. (Image credit: [Alc+18])

5.7.6 Adversarial Poses Were Not Found in ImageNet Classes via A Nearest-neighbor Search

We performed a nearest-neighbor search to check whether adversarial poses generated (in Section 5.3.1) can be found in the ImageNet dataset.

Retrieving Nearest Neighbors from A Single Class Corresponding to the 3D Object

We retrieved the five nearest training-set images for each adversarial pose (taken from a random selection of adversarial poses) using the fc7 feature space from a pre-trained AlexNet [KSH12]. The Euclidean distance was used to measure the distance between two fc7 feature vectors. We did not find qualitatively similar images despite comparing all ∼1,300 class images corresponding to the 3D object used to generate the adversarial poses (e.g., `cellphone`, `school bus`, and `garbage truck` in Figure 5.16, Figure 5.17, and Figure 5.18. This result supports the hypothesis that the generated adversarial poses are out-of-distribution.

Searching from the Validation Set

We also searched the entire 50,000-image validation set of ImageNet. Interestingly, we found the top-5 nearest images were sometimes from the same class as the *targeted* misclassification label (see Figure 5.24).

Figure 5.16: For each adversarial example (leftmost), we retrieved the five nearest neighbors (five rightmost photos) from all ∼1,300 images in the `cellular phone` class. The Euclidean distance between a pair of images was computed in the `fc7` feature space of a pre-trained AlexNet [KSH12]. The nearest photos from the class are mostly different from the adversarial poses. This result supports the hypothesis that the generated adversarial poses are out-of-distribution. The original, high-resolution figure is available at `https://goo.gl/X31VXh`. (Image credit: [Alc+18])

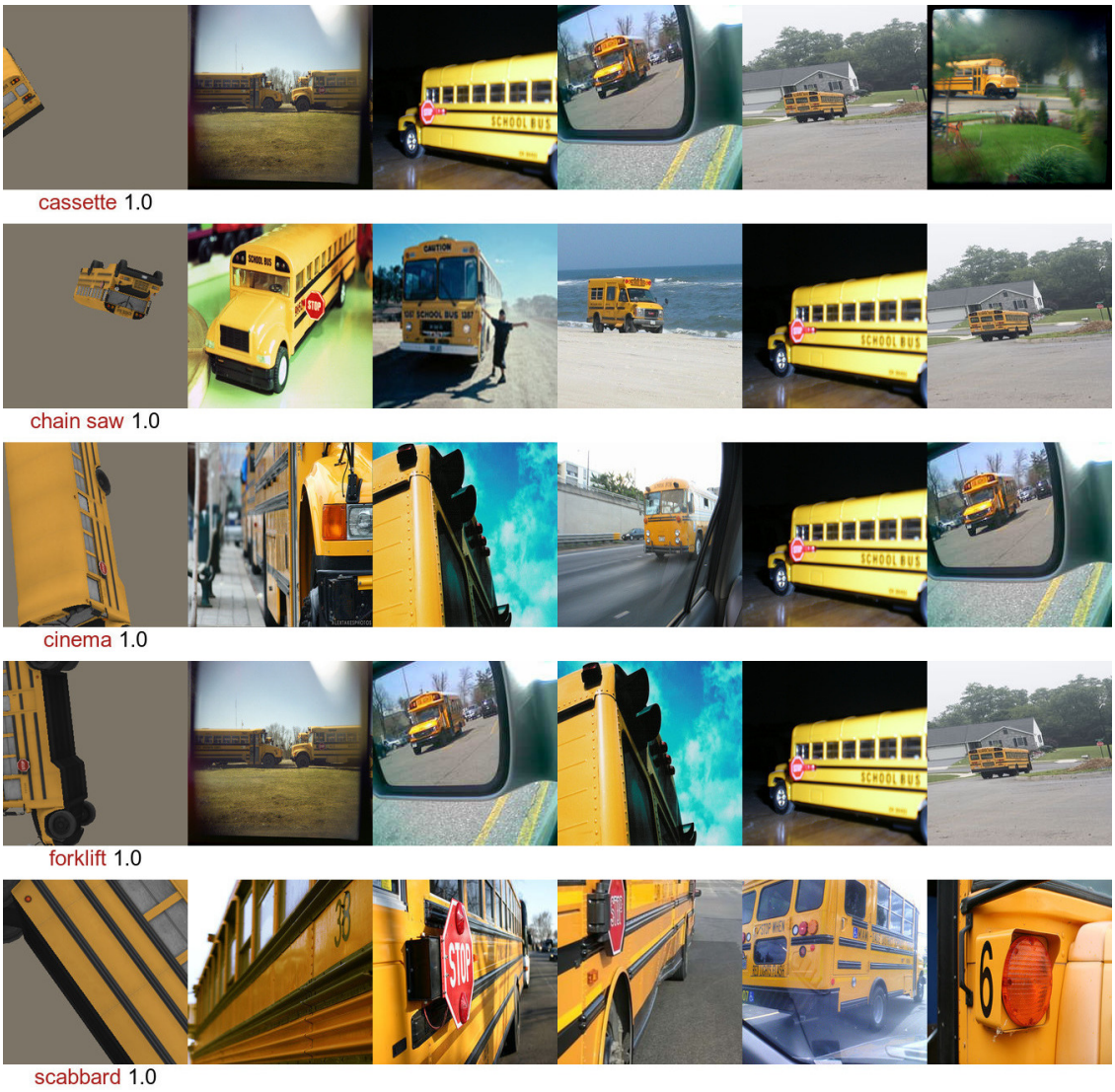


Figure 5.17: For each adversarial example (leftmost), we retrieved the five nearest neighbors (five rightmost photos) from all ∼1,300 images in the `school bus` class. The Euclidean distance between a pair of images was computed in the `fc7` feature space of a pre-trained AlexNet [KSH12]. The nearest photos from the class are mostly different from the adversarial poses. This result supports the hypothesis that the generated adversarial poses are out-of-distribution. The original, high-resolution figure is available at `https://goo.gl/X31VXh`. (Image credit: [Alc+18])

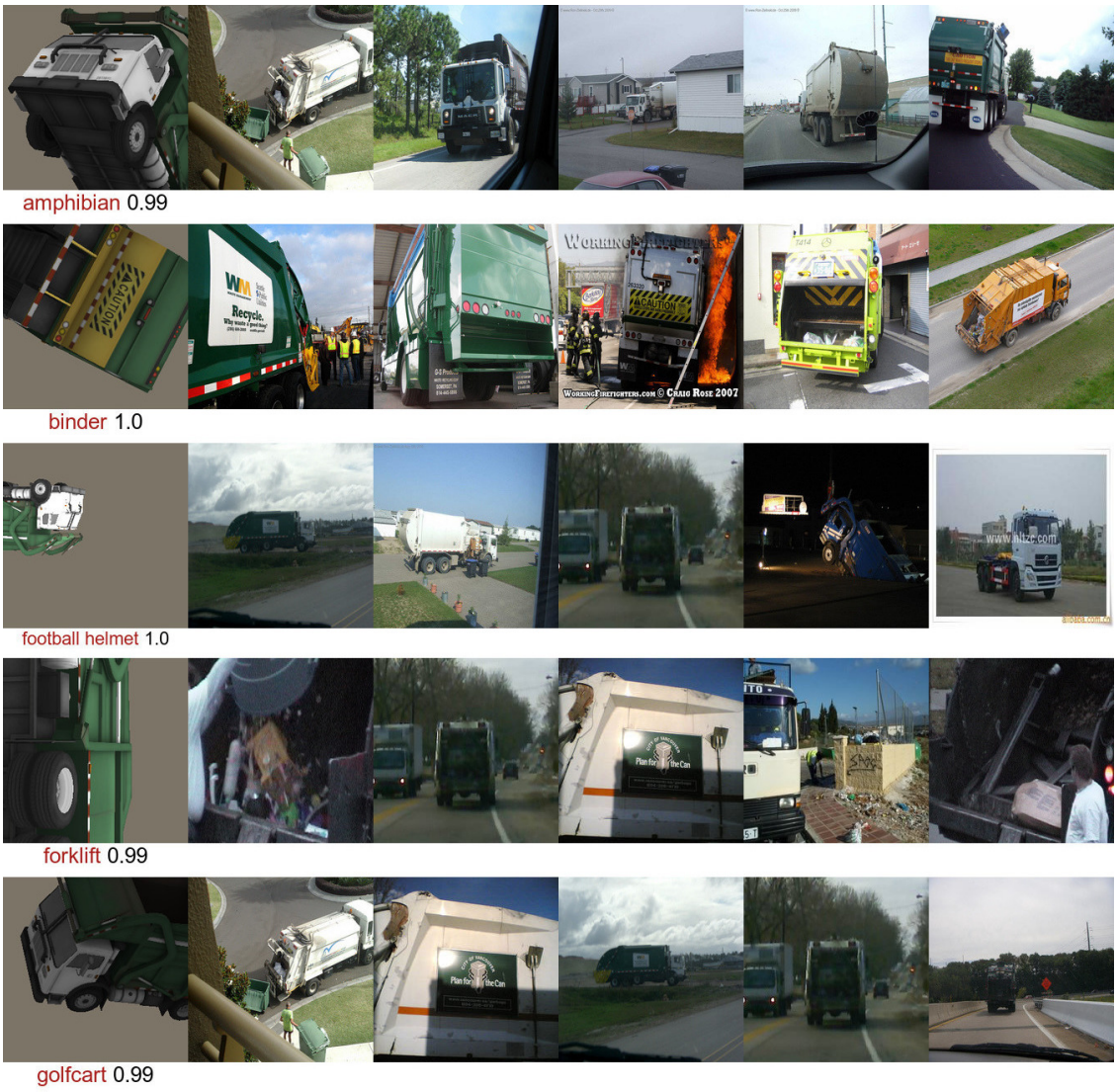


Figure 5.18: For each adversarial example (leftmost), we retrieved the five nearest neighbors (five rightmost photos) from all ∼1,300 images in the `garbage truck` class. The Euclidean distance between a pair of images was computed in the `fc7` feature space of a pre-trained AlexNet [KSH12]. The nearest photos from the class are mostly different from the adversarial poses. This result supports the hypothesis that the generated adversarial poses are out-of-distribution. The original high-resolution image is available at `https://goo.gl/X31VXh`. (Image credit: [Alc+18])

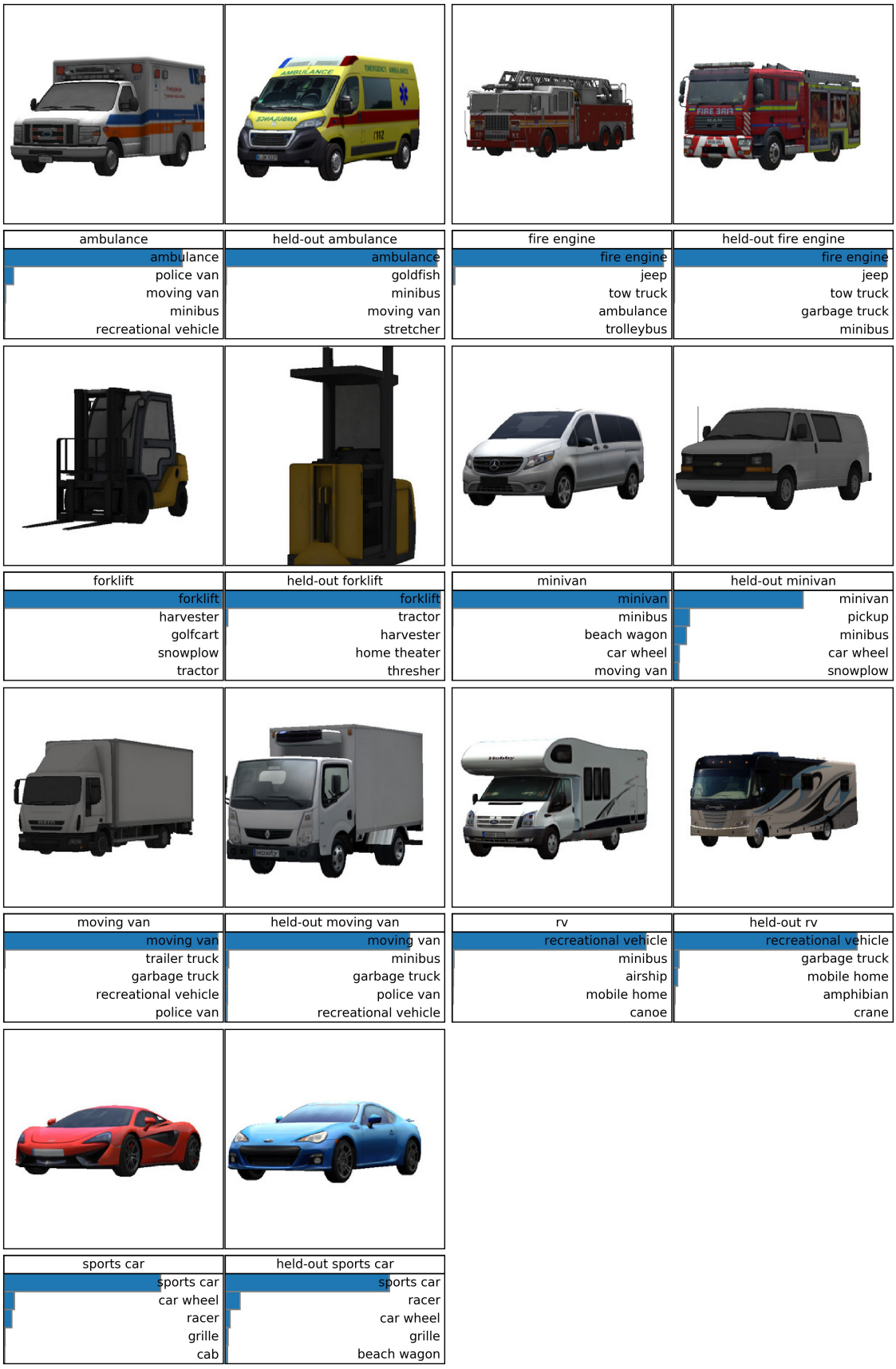
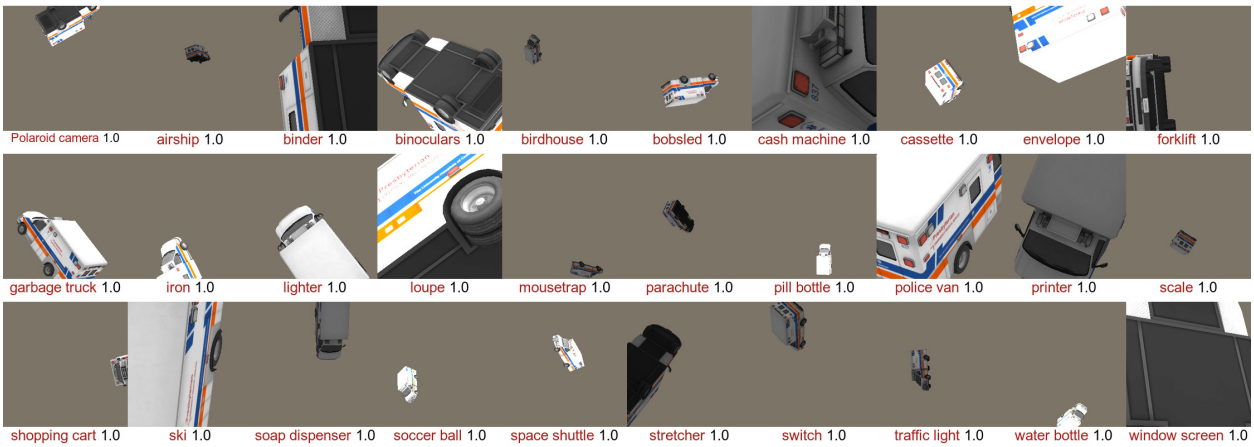
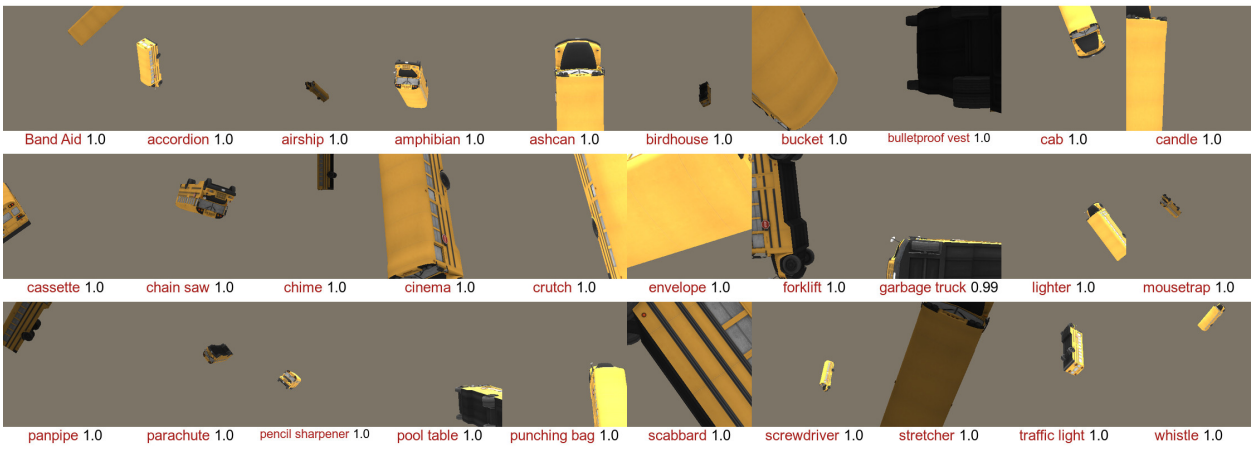


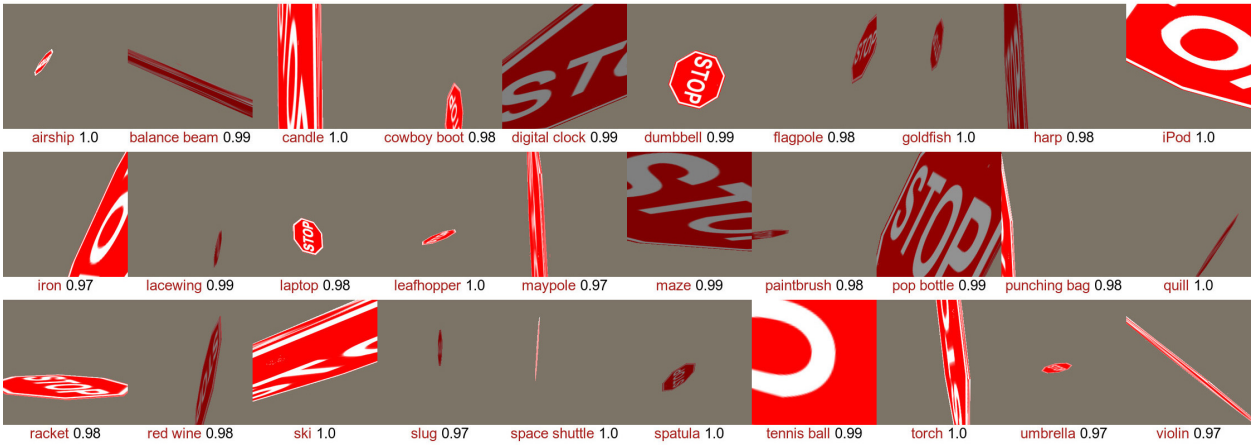
Figure 5.19: In Section 5.4, we trained an AlexNet classifier on the 1000-class ImageNet dataset augmented with 30 additional classes that contain adversarial poses corresponding to the 30 *known* objects used in the main experiments. We also tested this model on 7 *held-out* objects. Here, we show the renders of 7 pairs of (training-set object, held-out object). The



(a) `ambulance`



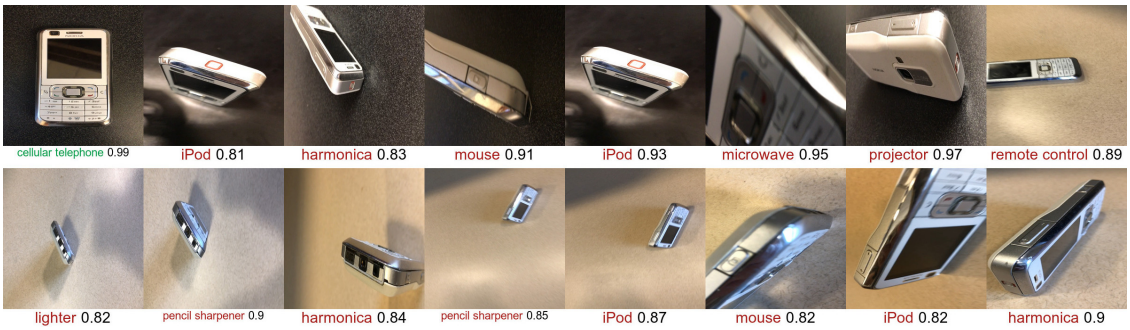
(b) `school bus`



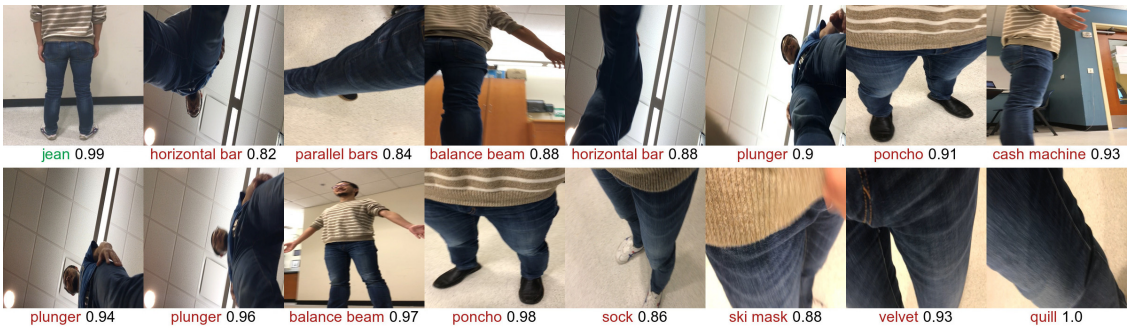
(c) `street sign`

Figure 5.20: 30 random adversarial examples misclassified by Inception-v3 [Sze+16] with high confidence ($p \geq 0.9$) generated from 3 objects: `ambulance`, `school bus`, and `street sign`. Below each image is the top-1 prediction label and confidence score. The original, high-resolution figures for all 30 objects are available at https://goo.gl/rvDzjy. (Image credit: [Alc+18])

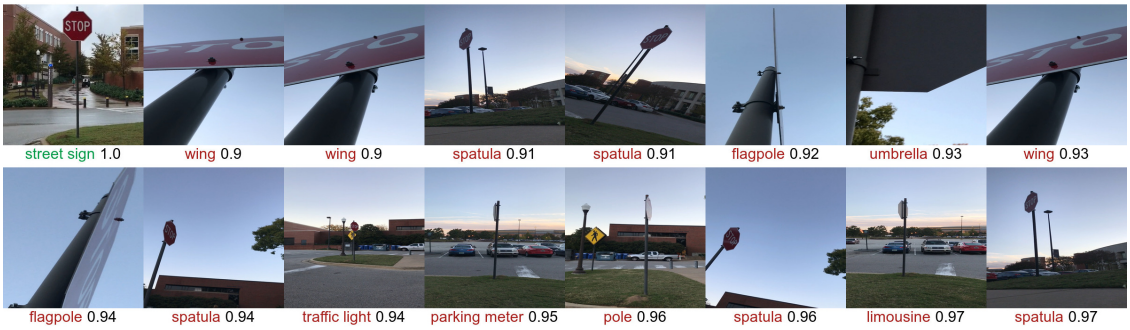
Figure 5.21: For each target class (e.g., `accordion piano`), we show five adversarial poses generated from five unique 3D objects. Adversarial poses are interestingly found to be homogeneous for some classes, e.g., `safety pin`. However, for most classes, the failure modes are heterogeneous. The original high-resolution figure is available at `https://goo.gl/37HYcE`. (Image credit: [Alc+18])



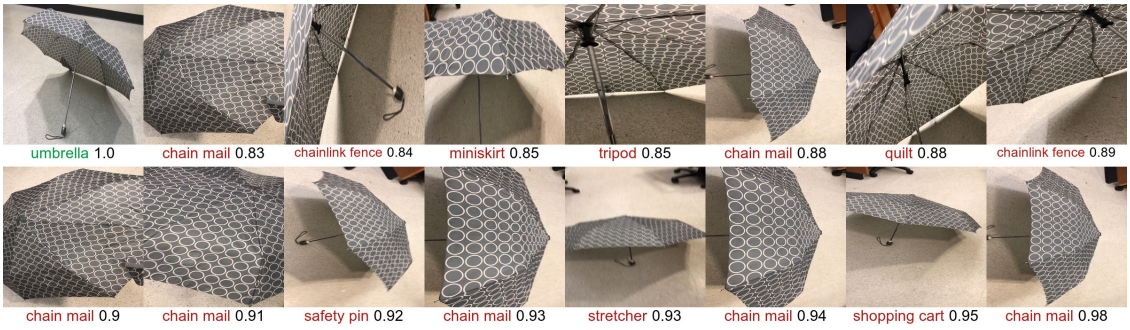
(a) `cellular phone`



(b) `jeans`



(c) `street sign`



(d) `umbrella`

Figure 5.22: Real-world, high-confidence adversarial poses can be found by taking photos from strange angles of a familiar object, here, `cellular phone`, `jeans`, `street sign`, and `umbrella`. While Inception-v3 [Sze+16] can correctly predict the object in canonical poses (the top-left image in each panel), the model misclassified the same objects in unusual poses. Below each image is its top-1 prediction label and confidence score. We took real-world videos of these four objects and extracted these misclassified poses from the videos. The original, high-resolution figures are available at https://goo.gl/zDWcjG. (Image credit: [Alc+18])

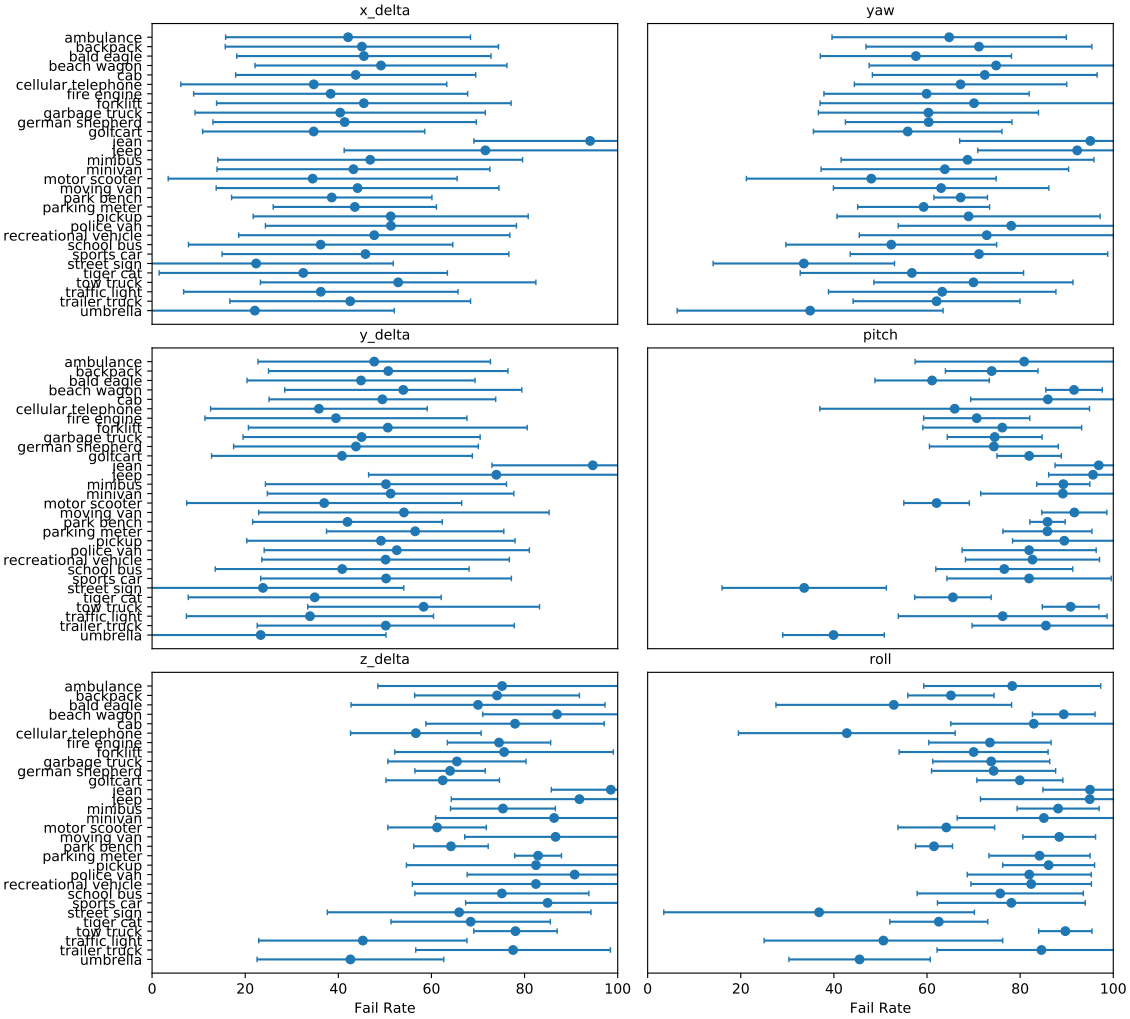


Figure 5.23: Inception-v3 [Sze+16] is sensitive to single parameter disturbances of object poses that had originally been correctly classified. For each object, we found 100 correctly classified 6D poses via a random sampling procedure (Section 5.3.3). Given each such pose, we re-sampled one parameter (shown on top of each panel, e.g., yaw) 100 times, yielding 100 classifications, while holding the other five pose parameters constant. In each panel, for each object (e.g., `ambulance`), we show an error plot for all resultant $100 \times 100 = 10,000$ classifications. Each circle denotes the mean misclassification rate (Fail Rate) for each object, while the bars enclose one standard deviation. Across all objects, Inception-v3 is more sensitive to changes in yaw, pitch, roll, and depth (z_{delta}) than spatial changes (x_{delta} and y_{delta}). (Image credit: [Alc+18])

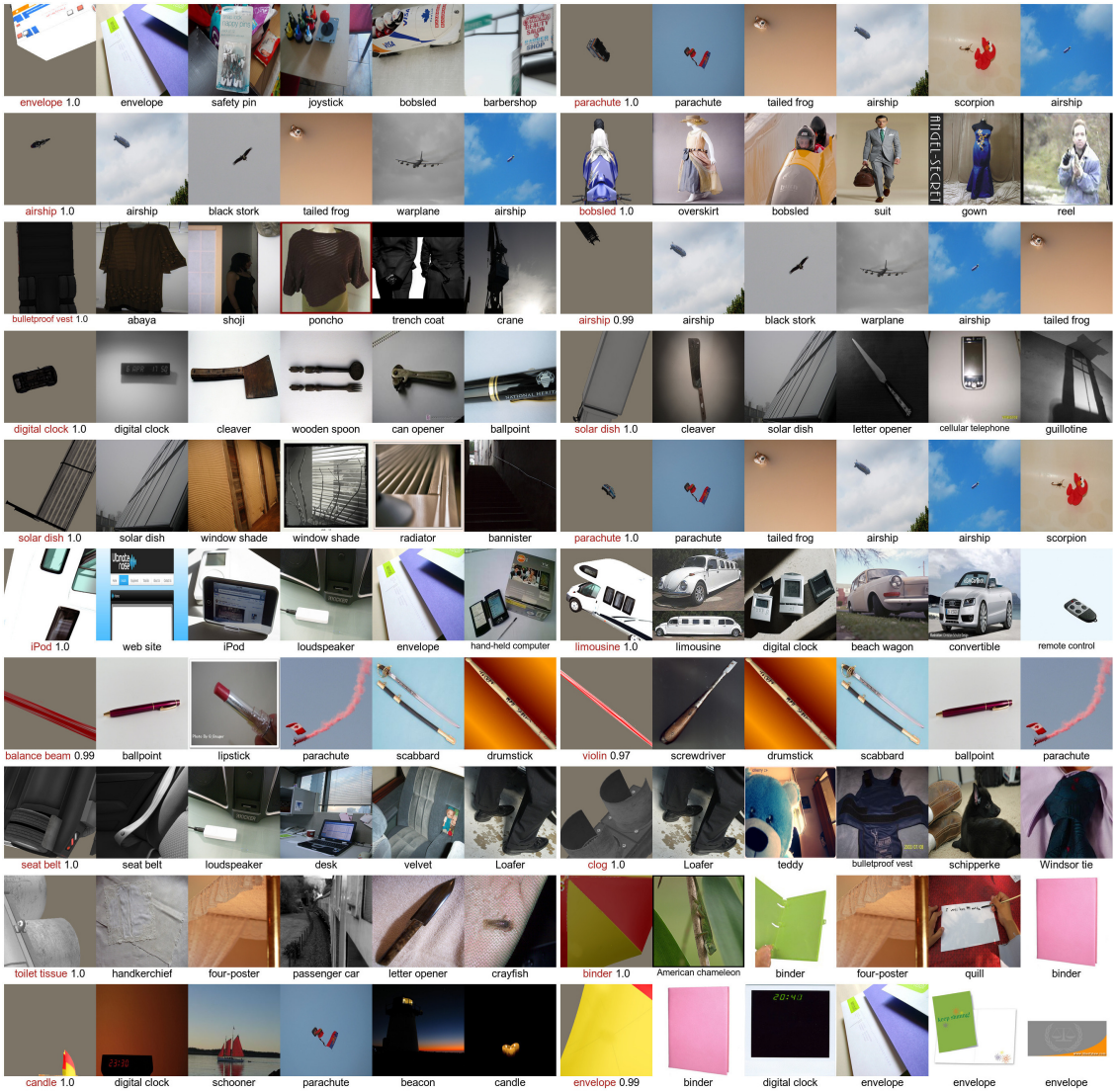


Figure 5.24: For each adversarial example (leftmost), we retrieved the five nearest neighbors (five rightmost photos) from the 50,000-image ImageNet validation set. The Euclidean distance between a pair of images was computed in the fc7 feature space of a pre-trained AlexNet [KSH12]. Below each adversarial example (AX) is its Inception-v3 [Sze+16] top-1 prediction label and confidence score. The associated ground-truth ImageNet label is beneath each retrieved photo. Here, we show an interesting, cherry-picked collection of cases where the nearest photos (in the fc7 feature space) are also qualitatively similar to the reference AX and sometimes come from the exact same class as the AX's predicted label. More examples are available at `https://goo.gl/8ib2PR`. (Image credit: [Alc+18])

Chapter 6

Conclusion

In this dissertation, we empirically study an emerging problem in machine learning community, i.e., the adversarial samples. In our first work, we propose to build a binary classifier to separate the adversarial samples from the clean one. This follows from the observation that the neural networks are sensitive to individual pixel values (which is exactly why we have adversarial samples in the first place). We demonstrate that the binary classifier approach could only successfully recognize the adversarial samples that follow similar distributions. Different the same adversarial methods with different hyper-parameters and different adversarial algorithms will usually generate adversarial samples that follow different distributions. In other words, the classifier may not generalize well to adversarial samples generated with a different method or the same method with different set of hyper-parameters. In our second work, we investigate the problem of generating text adversarial samples. We propose a simple yet effective framework to generate adversarial texts. In our framework, we first find potential adversarial texts in the embedding space, then use nearest neighbor to map back to the word space. The limitation of our framework is that we do not have a explicit way of controlling the quality of generated texts. Besides, the nearest neighbors search is a major computation bottleneck in our framework. In our third work, we show that adversarial samples for deep models exist in abundance in the natural world. Instead of pixel-hacking in 2D images, we are fooling the network by rendering 3D objects in *non-canonical* poses, e.g., slightly weird yaw-angle. This reveals a more severe problem since many of the poses really do exists in real world. We also show that the natural adversarial images transfer among different models.

Up to now, we mainly focus on a *descriptive*-style of exploration, i.e., we empirically show that the problem exists widely without giving an explicit explanation, nor a solution. Many work are along the lines of the *prescriptive*-style, seeking a rigorous explanation. We have seen some promising progress towards this direction, albeit slow due to the lack of understanding of deep models.

We think that this adversarial problem is only one piece in the jigsaw puzzle of a bigger picture, thus it is not solvable by itself, nor is it complete by itself. As a result, progress in other aspects may also help towards solving this adversarial problem.

In the end, I would like to conclude my dissertation borrowing an image from [Kar16]. The relates deeply with the bigger picture.

Figure 6.1: A joke that is not easily understood by machines [Kar16].

From this image, we can see where we are right now, and what is lacking in the current machine learning models.

What machine can do:

1. It can create a bounding box for each object it recognizes in the image.

2. It can precisely outline the (sub)component of each objects.

3. It can detect faces, and possibly smiley or not.

4. It can generate some description of the images.

5. It can generate images of similar styles.

6. etc.

What machine cannot do

1. It does not understand the concept of *mirror*, and that all the figures in the mirrors are *fake*.

2. It does not understand the concept of *shadows*.

3. it does not understand what the man on the scale is doing.

4. It does not understand the relationships among objects, figures, etc.

5. And the most important of all, it does not understand this joke. Mr. Obama is tipping his weight on the scale to play a trick on the man on the scale.

6. It does not understand ...

The observation is that despite the machine may finish many tasks (e.g., games playing, image recognition) at super-human level, it does not work the same way as we do. Adversarial examples demonstrate one facet of such disparity, e.g. most of the adversarial examples, that successfully trick machine learning model, do not interfere with human judgment. To understand and address this disparity, there may be a couple of interesting directions that I would like to follow as an extension to the work in this dissertation.

1. Interpretability. This line of research attempts to interpret the machine's reasoning in terms of human reasoning.

2. Learning representation. This line of research focus on how to represent the knowledge, and how to learn an efficient representation of the knowledge.

Bibliography

[Alc+18] Michael A. Alcorn et al. “Strike (with) a Pose: Neural Networks Are Easily Fooled By Strange Poses of Familiar Objects”. In: *CoRR* abs/1811.11553 (2018). arXiv: 1811.11553. URL: http://arxiv.org/abs/1811.11553.

[Alh+18] Hassan Abu Alhaija et al. “Geometric Image Synthesis”. In: *arXiv preprint arXiv:1809.04696* (2018).

[Ano19] Anonymous. “A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations”. In: *Submitted to International Conference on Learning Representations*. under review. 2019. URL: https://openreview.net/forum?id=BJfvknCqFQ.

[Ath+17] Anish Athalye et al. “Synthesizing Robust Adversarial Examples”. In: *CoRR* abs/1707.07397 (2017).

[BF17] Shumeet Baluja and Ian Fischer. “Adversarial Transformation Networks: Learning To Generate Adversarial Examples”. In: *CoRR* abs/1703.09387 (2017). URL: http://arxiv.org/abs/1703.09387.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O’Reilly Media, Inc.", 2009.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly Detection: a Survey”. In: *ACM computing surveys (CSUR)* 41.3 (2009), p. 15.

[Che+15] Chenyi Chen et al. "Deepdriving: Learning affordance for direct perception in autonomous driving". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2722–2730.

[CL92] Benjamin B Choi and Charles Lawrence. "Inverse Kinematics Problem in Robotics Using Neural Networks". In: *NASA Technical Memorandum* 105869 (1992), pp. 1–23. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.1730&rep=rep1&type=pdf.

[CW16] Nicholas Carlini and David Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *CoRR* abs/1608.04644 (2016). URL: http://arxiv.org/abs/1608.04644.

[CW18] N. Carlini and D. Wagner. "Audio Adversarial Examples: Targeted Attacks on Speech-To-Text". In: *ArXiv e-prints* (Jan. 2018). arXiv: 1801.01944 [cs.LG].

[Dom19] Szabolcs Dombi. *ModernGL - ModernGL 5.4.1 documentation*. https://moderngl.readthedocs.io/en/stable/index.html. (Accessed on 11/14/2018). 2019.

[Ebr+17] J. Ebrahimi et al. "HotFlip: White-Box Adversarial Examples for NLP". In: *ArXiv e-prints* (Dec. 2017). arXiv: 1712.06751 [cs.CL].

[Evt+17] I. Evtimov et al. "Robust Physical-World Attacks on Machine Learning Models". In: *ArXiv e-prints* (July 2017). arXiv: 1707.08945 [cs.CR].

[Gon+18] Zhitao Gong et al. "Adversarial Texts With Gradient Methods". In: *arXiv e-prints*, arXiv:1801.07175 (Jan. 2018), arXiv:1801.07175. arXiv: 1801.07175 [cs.CL].

[Goo+14] I. J. Goodfellow et al. "Generative Adversarial Networks". In: *ArXiv e-prints* (June 2014). arXiv: 1406.2661 [stat.ML].

[Goo+16] Ian Goodfellow et al. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[GPG17] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. "Learning to fly by crashing". In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE. 2017, pp. 3948–3955.

[Gra18] Henry Grabar. *The self-driving Uber that killed a pedestrian didn't brake. Here's why.* `https://slate.com/technology/2018/05/uber-car-in-fatal-arizona-crash-perceived-pedestrian-1-3-seconds-before-impact.html`. (Accessed on 07/13/2018). May 2018.

[GSS14] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and Harnessing Adversarial Examples". In: *ArXiv e-prints* (Dec. 2014). arXiv: `1412.6572 [stat.ML]`.

[GWK17] Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. "Adversarial and Clean Data Are Not Twins". In: *CoRR* abs/1704.04960 (2017).

[He+15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). URL: `http://arxiv.org/abs/1512.03385`.

[HG17] Dan Hendrycks and Kevin Gimpel. "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks". In: *Proceedings of International Conference on Learning Representations*. 2017.

[Hua+15] Ruitong Huang et al. "Learning With a Strong Adversary". In: *CoRR* abs/1511.03034 (2015). URL: `http://arxiv.org/abs/1511.03034`.

[HVD15] G. Hinton, O. Vinyals, and J. Dean. "Distilling the Knowledge in a Neural Network". In: *ArXiv e-prints* (Mar. 2015). arXiv: `1503.02531 [stat.ML]`.

[JAF16] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution". In: *European Conference on Computer Vision*. Springer. 2016, pp. 694–711.

[JL17] Robin Jia and Percy Liang. "Adversarial Examples for Evaluating Reading Comprehension Systems". In: *arXiv preprint arXiv:1707.07328* (2017).

[Kar16] Andrew Karparthy. "Connecting Images and Natural Language". Ph.D. dissertation. Stanford University, 2016. URL: https://cs.stanford.edu/people/karpathy/main.pdf.

[KB14] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). URL: http://arxiv.org/abs/1412.6980.

[KGB16a] A. Kurakin, I. Goodfellow, and S. Bengio. "Adversarial Examples in the Physical world". In: *ArXiv e-prints* (July 2016). arXiv: 1607.02533 [cs.CV].

[KGB16b] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial Machine Learning At Scale". In: *CoRR* abs/1611.01236 (2016). URL: http://arxiv.org/abs/1611.01236.

[Kim+15] Yoon Kim et al. "Character-Aware Neural Language Models". In: *arXiv preprint arXiv:1508.06615* (2015).

[Kim14] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *CoRR* abs/1408.5882 (2014).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[KUH18] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. "Neural 3D Mesh Renderer". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[Kus+15] Matt Kusner et al. "From word embeddings to document distances". In: *International Conference on Machine Learning*. 2015, pp. 957–966.

[KZG18] Danny Karmon, Daniel Zoran, and Yoav Goldberg. "Lavan: Localized and Visible Adversarial Noise". In: *arXiv preprint arXiv:1801.02608* (2018).

[Lam16] F Lambert. "Understanding the Fatal Tesla Accident on Autopilot and the Nhtsa Probe". In: *Electrek, July* (2016).

[LC10] Yann LeCun and Corinna Cortes. *MNIST handwritten digit database*. http://yann.lecun.com/e 2010. URL: http://yann.lecun.com/exdb/mnist/.

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. "Network in Network". In: *CoRR* abs/1312.4400 (2013). URL: http://arxiv.org/abs/1312.4400.

[Lia+17] Bin Liang et al. "Deep Text Classification Can Be Fooled". In: *arXiv preprint arXiv:1704.08006* (2017).

[Lin+14] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[Lin+17] Yen-Chen Lin et al. "Tactics of Adversarial Attack on Deep Reinforcement Learning Agents". In: *arXiv preprint arXiv:1703.06748* (2017).

[Liu+18] H.-T. D. Liu et al. "Adversarial Geometry and Lighting Using a Differentiable Renderer". In: *ArXiv e-prints* (Aug. 2018). arXiv: 1808.02651.

[Lu+17a] J. Lu et al. "NO Need To Worry About Adversarial Examples in Object Detection in Autonomous Vehicles". In: *ArXiv e-prints* (July 2017). arXiv: 1707.03501 [cs.CV].

[Lu+17b] J. Lu et al. "Standard Detectors Aren't (currently) Fooled By Physical Adversarial Stop signs". In: *ArXiv e-prints* (Oct. 2017). arXiv: 1710.03337 [cs.CV].

[Luo+15] Yan Luo et al. "Foveation-Based Mechanisms Alleviate Adversarial Examples". In: *CoRR* abs/1511.06292 (2015). URL: http://arxiv.org/abs/1511.06292.

[Maa+11] Andrew L. Maas et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational*

Linguistics: Human Language Technologies. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. URL: http://www.aclweb.org/anthology/P11-1015.

[Met+17] Jan Hendrik Metzen et al. "On Detecting Adversarial Perturbations". In: *arXiv preprint arXiv:1702.04267* (2017).

[MFF15] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "Deepfool: a Simple and Accurate Method To Fool Deep Neural Networks". In: *CoRR* abs/1511.04599 (2015). arXiv: 1511.04599. URL: http://arxiv.org/abs/1511.04599.

[Mik+13a] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and Their Compositionality". In: *CoRR* abs/1310.4546 (2013). URL: http://arxiv.org/abs/1310.4546.

[Mik+13b] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781 (2013). URL: http://arxiv.org/abs/1301.3781.

[Miy+15] Takeru Miyato et al. "Distributional Smoothing With Virtual Adversarial Training". In: *stat* 1050 (2015), p. 25.

[Mor78] Jorge J Moré. "The Levenberg-Marquardt algorithm: implementation and theory". In: *Numerical analysis*. Springer, 1978, pp. 105–116.

[MS15] Steve Marschner and Peter Shirley. *Fundamentals of computer graphics*. CRC Press, 2015.

[MYZ13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." In: *hlt-Naacl*. Vol. 13. 2013, pp. 746–751.

[Ngu+17] Anh Nguyen et al. “Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2017.

[NYC14] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. “Deep Neural Networks Are Easily Fooled: High Confidence Predictions for Unrecognizable Images”. In: *CoRR* abs/1412.1897 (2014). URL: http://arxiv.org/abs/1412.1897.

[OML05] Margarita Osadchy, Matthew L Miller, and Yann LeCun. “Synergistic Face Detection and Pose Estimation with Energy-Based Models”. In: *Advances in Neural Information Processing Systems*. 2005, pp. 1017–1024. ISBN: 9783540687948. DOI: 10.1007/11957959. URL: https://doi.org/10.1007/11957959.

[Pap+15a] Nicolas Papernot et al. “Distillation As a Defense To Adversarial Perturbations Against Deep Neural Networks”. In: *CoRR* abs/1511.04508 (2015). URL: http://arxiv.org/abs/1511.04508.

[Pap+15b] Nicolas Papernot et al. “The Limitations of Deep Learning in Adversarial Settings”. In: *CoRR* abs/1511.07528 (2015). URL: http://arxiv.org/abs/1511.07528.

[Pec+17] Jonathan Peck et al. “Lower bounds on the robustness to adversarial perturbations”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 804–813. URL: http://papers.nips.cc/paper/6682-lower-bounds-on-the-robustness-to-adversarial-perturbations.pdf.

[PMG16] N. Papernot, P. McDaniel, and I. Goodfellow. “Transferability in Machine Learning: From Phenomena To Black-Box Attacks Using Adversarial Samples”. In: *ArXiv e-prints* (May 2016). arXiv: 1605.07277 [cs.CR].

[Pre18] Associated Press. *Tesla that crashed into truck was on Autopilot*. 2018. URL: https://nypost.com/2018/05/14/tesla-that-crashed-into-truck-was-on-autopilot (visited on 01/11/2018).

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[PyT18] PyTorch. *PyTorch Master Documentation*. https://pytorch.org/docs/stable/torchvision/models.html. (Accessed on 11/14/2018). 2018.

[RF18] Joseph Redmon and Ali Farhadi. "Yolov3: an Incremental Improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning Representations By Back-Propagating Errors". In: *nature* 323.6088 (1986), p. 533.

[ŘS10] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. "The Earth Mover's Distance As a Metric for Image Retrieval". In: *International journal of computer vision* 40.2 (2000), pp. 99–121.

[Rus+15] Olga Russakovsky et al. "Imagenet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[Sam+18] Carlos Sampedro et al. "A Fully-Autonomous Aerial Robot for Search and Rescue Applications in Indoor Environments Using Learning-Based Techniques". In: *Journal of Intelligent & Robotic Systems* (2018), pp. 1–27.

[Sch+13] Walter J Scheirer et al. “Toward Open Set Recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.7 (2013), pp. 1757–1772.

[SGS15] Rupesh K Srivastava, Klaus Greff, and Juergen Schmidhuber. “Training Very Deep Networks”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 2377–2385. URL: http://papers.nips.cc/paper/5850-training-very-deep-networks.pdf.

[Shr+16] Ashish Shrivastava et al. “Learning From Simulated and Unsupervised Images Through Adversarial Training”. In: *CoRR* abs/1612.07828 (2016).

[SLS+17] Masashi Sugiyama, Neil D Lawrence, Anton Schwaighofer, et al. *Dataset shift in machine learning*. The MIT Press, 2017.

[SM17] Suranjana Samanta and Sameep Mehta. “Towards Crafting Text Adversarial Samples”. In: *arXiv preprint arXiv:1707.02812* (2017).

[Suc+17] Felipe Petroski Such et al. “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning”. In: *arXiv preprint arXiv:1712.06567* (2017).

[SVK17] J. Su, D. Vasconcellos Vargas, and S. Kouichi. “One Pixel Attack for Fooling Deep Neural networks”. In: *ArXiv e-prints* (Oct. 2017). arXiv: 1710.08864 [cs.LG].

[Sze+13] Christian Szegedy et al. “Intriguing Properties of Neural Networks”. In: *CoRR* abs/1312.6199 (2013). URL: http://arxiv.org/abs/1312.6199.

[Sze+16] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[TE11] Antonio Torralba and Alexei A Efros. "Unbiased look at dataset bias". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, pp. 1521–1528.

[Tia+18] Yuchi Tian et al. "Deeptest: Automated testing of deep-neural-network-driven autonomous cars". In: *Proceedings of the 40th International Conference on Software Engineering*. ACM. 2018, pp. 303–314.

[Tim16] New York Times. *Tesla car on autopilot crashes, killing driver, United States News & Top Stories - The Straits Times*. `https://www.straitstimes.com/world/united-states/tesla-car-on-autopilot-crashes-killing-driver`. (Accessed on 06/14/2018). July 2016.

[WG16] D Warde-Farley and I Goodfellow. "Adversarial Perturbations of Deep Neural Networks". In: *Perturbation, Optimization and Statistics*. Ed. by Tamir Hazan, George Papandreou, and Daniel Tarlow. 2016.

[Wil92] Ronald J Williams. "Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning". In: *Reinforcement Learning*. Springer, 1992, pp. 5–32.

[Won17] C. Wong. "DANCin Seq2seq: Fooling Text Classifiers With Adversarial Text Example Generation". In: *ArXiv e-prints* (Dec. 2017). arXiv: `1712.05419 [cs.LG]`.

[Woo+99] Mason Woo et al. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[Xia+16] Yu Xiang et al. "Objectnet3d: A large scale database for 3d object recognition". In: *European Conference on Computer Vision*. Springer. 2016, pp. 160–176.

[Xia+18] C. Xiao et al. "Generating Adversarial Examples With Adversarial Networks". In: *ArXiv e-prints* (Jan. 2018). arXiv: `1801.02610 [cs.CR]`.

[Yua+17] Xiaoyong Yuan et al. "Adversarial Examples: Attacks and Defenses for Deep Learning". In: *arXiv preprint* (2017). DOI: arXivpreprintarXiv:1712.07107. URL: https://doi.org/arXiv%20preprint%20arXiv:1712.07107.

[ZDS17] Z. Zhao, D. Dua, and S. Singh. "Generating Natural Adversarial Examples". In: *ArXiv e-prints* (Oct. 2017). arXiv: 1710.11342 [cs.LG].

[ZL16] Barret Zoph and Quoc V. Le. "Neural Architecture Search With Reinforcement Learning". In: *CoRR* abs/1611.01578 (2016). URL: http://arxiv.org/abs/1611.01578.

[ZXY16] Zhuotun Zhu, Lingxi Xie, and Alan L Yuille. "Object Recognition With and Without Objects". In: *arXiv preprint arXiv:1611.06596* (2016).

[ZZL15] Xiang Zhang, Junbo Zhao, and Yann LeCun. "Character-level Convolutional Networks for Text Classification". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 649–657. URL: http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf.