# Reproducing Neural Network Spoofing Methods

Jason Anderson
Stanford University
jand271@stanford.edu

Cheng Chang
Stanford University
chc012@stanford.edu

Sanchit Jain
Stanford SCPD
sanchit@stanford.edu

## Abstract

*Many Convolutional Neural Networks (CNNs) architectures, such as ResNet, provide limited robustness guarantees, and may be vulnerable to attacks that significantly degrade their performance. We investigate methods that attack CNNs used for image classification and object detection by minimally changing input images and creating adversarial samples. We experiment on and compare previously proposed methods of creating adversarial samples, including the Adversarial Patch method (AG), Single Pixel Attack (SPA), and the Fast Gradient Sign Method (FGSM). These experiments will hopefully help us find ways of using Generative Adversarial Networks (GANs) to create adversarial samples based on images in multiple data sets.*

## 1. Introduction

There is a broad range of literature on methods to attack neural networks with adversarial examples. These methods can be categorized into black-box and white-box attacks. White-box attacks requires knowledge about the defender model, such as its training process or loss functions. A black-box attack would work without this knowledge.

The Adversarial Patch method (AP), proposed by [1], can significantly degrade the performance of many CNN-based image classification models by simply overlaying a small but highly unusual patch, created based on the image of a toaster, to the input images. [1] suggests that AP is robust to many CNN-based classification models, including those with ResNet, Inception, and VGG architectures. Though AP makes visible change to the input image, it is powerful in that it is a black-box attack.

Proposed by Su, et al in [4], One pixel attack (a.k.a. single pixel attack) causes neural networks to misclassify images while being limited by an extreme constraint of modifying only a single pixel of the image. It was . It is a semi black-box attack as it does not need any other information of the model except the final probability labels of each class.

Another method is called the Fast Gradient Sign Method ("FGSM") [2]. FGSM is remarkably easy to to compute and has a high probability of success. FGSM linearizes the neural network cost function to form a convex problem. The convex problem finds the adversarial image that maximizes the cost function subject to an infinity norm constraint that limits the deviation of the adversarial image to the original image.

Figure 1. The adversarial "toaster".

As a first step, we experiment with the three methods over standard image classification models from well-established online model libraries. We will use the result of these experiments to understand the best approach to create a GAN model that conduct black-box attack to a variety of image classification and object detection models.

### 1.1. Data and Frameworks

For our project, we will demonstrate our methods with ImageNet, but leave the option to apply our methods to a different dataset after showing success with classification models trained over ImageNet. For baseline experiments, we use different ResNet models. They require image inputs with RGB channels and a size of 224 by 224. We used a library called Pytorch Image Models (timm) to download and use pre-trained CNN-based image classification models, along with the required image transformation [5]. We now describe our experiment with the baseline attack methods in the following sections.

## 2. Adversarial Patch

As mentioned in Section 1, [1] suggests that the adversarial patch can be easily applied to any image and create

good attack result. We investigate the proper way of applying the patch in this section.

### 2.1. Apply Directly to Image

In this experiment, we resize the patch to about $4\%$ ($\frac{1}{25}$) of the size of the input images, put the patch at the top left corner of the image, and check the differences between the top 5 labels that a ResNet-50 model applies to the original and patched images. To our surprise, the model appears to be very robust to this attack method. There is no significant degradation of confidence between the two results, as the model outputs the correct label ("boxer") with over $99\%$ confidence for both images. This result also stays consistent after we experiment with different images, location of applying the patches over the images, and patch size.

### 2.2. Apply after Image Transformation

Instead of applying the patch to the raw image directly, we find that applying the patch to the transformed image before it is feed into the model works very well as an attack. With the same patch size (4%), image ("boxer"), and patch location (top-left corner), the post-transformation patching causes the model performance to degrade significantly, with the probability of the correct label very close to 0. This result also holds with other underlying images and random patch locations. The attack strength over the classification accuracy become insignificant only after the patch size is reduced to about 1% of the input image.

Contrary to the results reported by the original AP paper, the spoofed model does not report "toaster" as the most probable label for the adversarial images. Instead, it typically reports labels within similar categories as the original label. For example, the most probable label of the "boxer" image becomes "kelpie", another breed of dog, or "dhole", a fox-like animal. This is expected behavior since the patch now looks nothing like toaster now, but it also suggests that the new method produces less significant errors than what the original paper suggests.

## 3. Single Pixel Attack

[4] states that generating adversarial images can be formalized as an optimization problem with constraints. An input image can be represented by a vector in which each scalar element represents one pixel. Let **f** be the target image classifier which receives n-dimensional inputs, **x = (x1, .., xn)** be the original image correctly classified with class t. The probability of x belonging to the class t is therefore **f_t(x)**. The vector **e(x) = (e1, .., en)** is an additive adversarial perturbation according to **x**, the target class *adv* and the limitation of maximum modification L. Note that L is always measured by the length of vector e(x). The goal of adversaries in the case of targeted attacks is to find the op-

Figure 2. Top: original image classified correctly as a Boxer. Middle: image with pre-transformation patching (size: 4%) classified correctly as a Boxer. Bottom: image with post-transformation patching (size: 4%) classified as a Kelpie, with the probability assigned to Boxer degrading from over 0.997 to around $10^{-6}$.

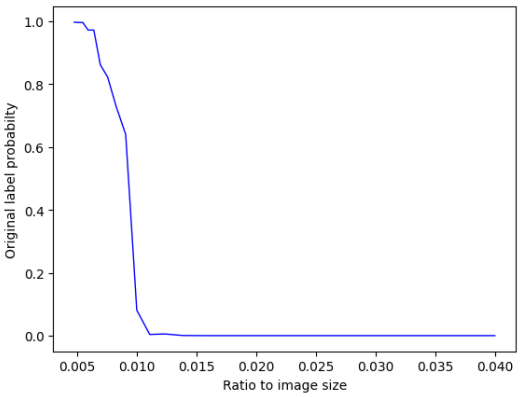


Figure 3. Relation between post-transformation patch size and original label probability. For this particular example, the correct label ("boxer") is reduced to 0 if the patch occupies about 1% of the image.

timized solution for the following question:

$$maximize_{e(x)} f_{\text{adv}}(x + e(x))$$
$$\text{subject to} \quad ||e(x)||_0 \leq L$$

The problem involves finding two values: (a) which dimensions that need to be perturbed and (b) the corresponding strength of the modification for each dimension. In their approach, the equation is slightly different:

$$maximize_{e(x)} f_{\text{adv}}(x + e(x))$$
$$\text{subject to} ||e(x)||_0 \leq d$$

where d is a small number. In the case of one-pixel attack d = 1. Overall, few-pixel attack conducts perturbations on the low-dimensional slices of input space. In fact, one-pixel perturbation allows the modification of an image towards a chosen direction out of n possible directions with arbitrary strength.

Let's discuss the problem of finding the pixels that will result in a successful attack - first, formulate it as an optimization problem: in an untargeted attack, minimize the confidence of the correct class, and in a targeted attack, maximize the confidence of a target class.

When performing black-box optimizations such as the one pixel attack, it can be very difficult to find an efficient gradient-based optimization that will work for the problem. It would be nice to use an optimization algorithm that can find good solutions without relying on the smoothness of the function. In our case, we have discrete integer positions ranging from 0 to 31 and color intensities from 0 to 255, so the function is expected to be jagged. For this purpose, we use an algorithm called differential evolution.

Differential evolution ([3]) is a type of evolutionary algorithm where a population of candidate solutions generate offspring which compete with the rest of the population each generation according to their fitness. Each candidate solution is represented by a vector of real numbers which are the inputs to the function we would like to minimize. The lower the output of this function, the better the fitness. The algorithm works by initializing a (usually random) population of vectors, generating new offspring vectors by combining (mutating) individuals in the population, and replacing worse-performing individuals with better candidates.

In the context of the one pixel attack, our input will be a flat vector of pixel values:

$$X = (x_1, y_1, r_1, g_1, b_1, x_2, y_2, r_2, g_2, b_2, ...) \quad (1)$$

These will be encoded as floating-point values, but will be floored back into integers to calculate image perturbations. First we generate a random population of perturbations.

$$\mathbf{P} = (X_1, X_2, \ldots, X_n) \quad (2)$$

Then, on each iteration we calculate new mutant children using the formula

$$X_i = X_{r1} + F(X_{r2} - X_{r3}) \quad (3)$$

such that

$$r1 \neq r2 \neq r3 \quad (4)$$

where r1,r2,r3 are random indices into our population **P**, and **F = 0.5** is a mutation parameter. Basically, we pick 3 random individuals from the previous generation and recombine them to make a new candidate solution. If this candidate gives a lower minimum at position (i.e., the attack is closer to success), replace the old with this new one. This process repeats for several iterations until our stopping criterion, which is when we find an image that successfully completes the attack.

With the CIFAR-10 dataset, whose images are of the size 32x32, we had more success than with the ImageNet dataset, whose images have larger dimensions.

## 4. Fast Gradient Sign Method

In the simplest form of the Fast Gradient Sign Method, we perturb an existing input to cause the network to misclassify[2]. The perturbation is usually imperceptible to the human eye.

Let an input image be $x$ and its true label $y$. Let the neural network parameters be $\theta$ and the loss function by which the neural network is trained be $J(\theta, x, y)$. The Fast Gradient Sign Method maximizes the loss function (linearized) on an new adversarial example $J(\theta, x^{\text{adversarial}}, y)$ while constraining the perturbation to the example input. To do this, they linearize $J$, and form the following convex optimization problem.

$$\arg\max_x = J(\theta, x, y) + (x^{\text{adversarial}} - x)\nabla_x J(\theta, x, y)$$
$$\text{s.t.} \quad ||x^{\text{adversarial}} - x||_\infty < \epsilon$$

The solution this optimization problem is the following.

$$x^{\text{adversarial}} = x + \epsilon \cdot \text{sign}\left(\nabla_x J(\theta, x, y)\right)$$

. The relevant gradient can be efficiently computed via backpropagation.

### 4.1. Modifying FGSM

We modified the original FGSM problem to allow us to get better adversarial examples and spoof a specific label specified by a user. To get better images, we implemented a stepping loop that would perturb the input image in small steps. In the below equation, $\epsilon$ functions as a learning rate.

Figure 4. Original image (Top) classified correctly as a Boxer and FGSM-perturbed image (bottom) classified as American Bull Dog.

Figure 5. Original image (Top) classified correctly as Boxer and FGSM step perturbed image (bottom) classified as American Bull Dog via the step method of Section 4.1.

$$x \leftarrow x + \epsilon \cdot \text{sign}\left(\nabla_x J(\theta, x, y)\right)$$

From there, we repeatedly apply this update rule until changing the image classification.

To have an image be misclassified to an arbitrary label, we use the aforementioned step method, but replace the cost function with the score output the label we wish the adversarial example to have $J(\theta, x, y) = y_{\text{target}}$.

### 4.2. Experiments

We used a pre-trained ResNet resnet50-oxford-iiit-pet which classifies a variety of species of dogs. We used a test image downloaded from google images with no relation to the model. In Figure 4, a boxer image downloaded from the internet is classified correctly but then modified via FGSM to be misclassified into an American Bulldog. The image differences are obvious to the human eye.

Figure 5, the same boxer image is put through the stepping method of Section 4.1. In addition to finding adversarial examples with closer differences, we are able to similarly give an image an arbitrary classification.

## 5. Conclusion

For our Project Milestone, we found the Python tools, data and models to move forward on our project. We implemented some baseline methods, provided some preliminary extensions, and performed some preliminary experiments.

## References


[1] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer. Adversarial patch, 2018.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2015.

[3] R. S. . K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, 1997.

[4] J. Su, D. V. Vargas, and K. Sakurai. One pixel attack for fooling deep neural networks, 2018.

[5] R. Wightman. Pytorch image models.