ADVERSARIAL EXAMPLES

&

OBFUSCATED GRADIENTS

project for:
Neural Network exam

FRANCESCO PERACCHIA MATRICOLA 1906895

Index

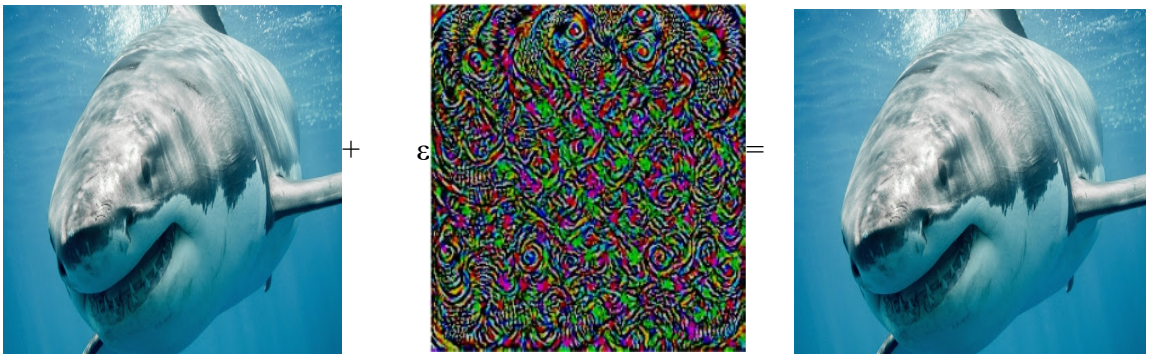
1. Introduction

Recent developments have shown that particular types of gradients leads to a false sense of security in defenses against Adversarial examples. This discussion will start with a short recap on what is an Adversarial examples

1.1 Adversarial examples

"Adversarial examples are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake; they're like optical illusions for machines"

General idea beyond adversarial examples can be explained whit this case, suppose that an image of a specific class, like a generic animal, is correctly classified with high confidence, the attacker adds a small perturbation that has been chosen to make the image be recognized with high confidence in a wrong class.



Shark 70% confidence Perturbation Red fish 95% confidence

These is not only a theoretical problem, is real and very dangerous for most Machine Learning applications, like in autonomous driving by using stickers or paint to create an adversarial stop sign that the vehicle would interpret as a 'yield' or other sign.

White-box vs. Black-box: Adversarial example generation methods can be classified into white-box and black-box generation methods. White-box generation methods require classifier model to be known. Black-box generation methods assume that given an input only output of the classifier is available.

One-shot vs. Iterative: Another important classification is how our algorithms solve the optimization problem. In one-shot methods, the optimization problem is solved using a one-shot optimization method. In iterative methods an iterative optimization method such as gradient descent is used to solve the same problem, generally one-shot methods can generate examples fast but iterative methods are more robust to defense methods.

1.2 Why is so difficult solve this problem ?

It is difficult to construct a theoretical model of the adversarial example crafting process, hence Adversarial examples are a concrete problem and it's very hard to defend against their.
For many ML models, adversarial examples are solutions to an optimization problem that is non-linear and non-convex.
It's very hard to make a generalization that a defense that will rule out a set of adversarial examples, this because there aren't good theoretical tools for describing the solutions to these complicated optimization problems.
Adversarial examples are also hard to defend against because they require machine learning models to produce good outputs for every possible input and machine learning models work very well but only work on a very small amount of all the many possible inputs they might find.
Designing a defense that can protect against a powerful, adaptive attacker is an important research area, not adaptive solution aren't useful, today the approach is find a handcrafted defense for each different type of problem.

1.3 Basics Definitions

A neural network can be defined with $f(\cdot)$, we can use this NN for solve classification tasks, the probability that image *x* corresponds to label *i* is represented here $f(x)_i$
We classify images, represented as $x \in [0, 1]^{w \cdot h \cdot c}$ for a *c-channel* image of width *w* and height *h*, hence $f_j(\cdot)$ to refer to layer j of the neural network, and $f_{1..j}(\cdot)$ the composition of layers from 1 to j, denote the classification of the network as $c(x) = \arg\max_i f(x)_i$
,and $c^*(x)$ denotes the true label.
After this short introduction, it's necessary give a formal definition for adversial example
Given an image x and classifier $f(\cdot)$, an adversarial example x' must grant **two properties**:

- $D(x, x')$ is small for some distance metric D
- $c(x') \neq c^*(x)$, x and x' are graphically similar but one is classified incorrectly

Hence generating AE, can be seen as an optimization problem, where it's possible do gradient ascent on the input space through the direction of the perturbation to get adversarial examples, in a similar way to the gradient descent on the parameters of a model.

1.4 Measure Similarity Distortion

To measure similarity between images, there are many metrics, two possible ways to measure similarity are l_{00} and l_2 Between two images which have a small distortion under either of these metrics will appear visually identical, l_{00} distance are in normalized [0, 1] space, a distortion of 0.031 corresponds to 8/256, and l_2 distance as the total root-mean-square distortion normalized by the total number of pixels.

[1][2]

2. Obfuscated Gradients

A possible definition for gradient masking defense is this:

"A defense is said to cause gradient masking if it "does not have useful gradients" for generating adversarial examples."

In few words gradient masking is known to be an incomplete defense to adversarial examples. Despite this already known definition, many ICLR 2018 defenses suffer from this effect.
It's necessary find the way by these defenses causes obfuscated gradients, hence we want find how these learn to break gradient descent, learning to make the gradients point the wrong direction.
The constructed defense can causes gradient masking as obfuscated gradients, so it's reported a classification that list in three ways defenses obfuscate gradients.

Shattered gradients are caused when there is non-differentiable defense, so introduces numeric instability, or otherwise causes a gradient to be nonexistent or incorrect. Defenses of this type can do this unintentionally, by using differentiable operations.

Stochastic gradients are caused by randomized defenses, where either the network itself is randomized or the input is randomly transformed before being fed to the classifier, causing the gradients to become randomized.

Exploding & Vanishing gradients are often caused by defenses that consist of multiple iterations of neural network evaluation, feeding the output of one computation as the input of the next. This type of computation, when unrolled, can be viewed as an extremely deep neural network evaluation, which can cause vanishing/exploding gradients.

2.1 Identifying Obfuscated & Masked Gradients

Some defenses intentionally break gradient descent and cause obfuscated gradients while others defenses unintentionally break gradient descent. The cause of gradient descent being broken is in the design of neural network itself.
There are clues that although they do not guarantee proof, they are possible indicators of obfuscated gradient:

- **One-step attacks perform better than iterative attacks**
- **Black-box attacks are better than white-box attacks**
- **Unbounded attacks do not reach 100% success.**
- **Random sampling finds adversarial examples.**
- **Increasing distortion bound does not increase success**

Iterative optimization-based attacks applied in a **white-box** setting are strictly stronger than single-step attacks and should give strictly superior performance. If single-step methods give performance superior to iterative methods, it is likely that the iterative attack is becoming stuck in its optimization search at a local minimum.

The **black-box** threat model is a strict subset of the **white-box** threat model, so attacks in the white-box setting should perform better; if a defense is obfuscating gradients, then black-box attacks (which do not use the gradient) often perform better than white-box attacks.

With unbounded distortion, any classifier should have 0% robustness to attack.
If an attack does not reach 100% success with sufficiently large distortion bound, this indicates the attack is not performing optimally against the defense, and the attack should be improved.

Brute force random search within some ε ball should not find adversarial examples when gradient-based attacks do not.

A larger distortion bound should monotonically increase attack success rate; significantly increasing distortion bound should result in significantly higher attack success rate.

2.2 Shattered

Shattered gradients, caused either **unintentionally**, (e.g. by numerical instability), or **intentionally**, (by using non-differentiable operations) result in nonexistent or incorrect gradients. To attack defenses, is possible introduce a technique called **Backward Pass Differentiable Approximation** (BPDA). Is possible use a special estimator called "**Straight-Through Estimator**"
Definition:

- Given a pre-trained classifier $f(\cdot)$
- Construct a preprocessor $g(\cdot)$
- Let the secured classifier $f'(x) = f(g(x))$
- $g(x) = x$ such g(·) may perform image denoising to remove the adversarial perturbation

"If g(·) is smooth and differentiable, then computing gradients through the combined network ˆf is often sufficient to circumvent thedefense"

Problem:
Recently functions $g(\cdot)$ are neither smooth nor differentiable, and therefore can not be backpropagated through to generate adversarial examples, with a white-box attack that requires gradient signal.

Solution:
Because g is constructed with the property that $g(x) \approx x$, we can approximate its derivative as the derivative of the identity function: $\nabla_x g(x) \approx \nabla_x x = 1$. Therefore, we can approximate the derivative of $f(g(x))$ at the point x' as: $\nabla_x f(g(x)) \approx \nabla_x f(x), with\ x = x', x = g(x')$
This allows us to compute gradients and therefore mount a white-box attack.

This approximation of the true gradient is useful when averaged over many iterations of gradient descent still generates an adversarial example.

Full Attack generalization (BPD)

The above attack is useful for a simple class of networks expressible as f(g(x)) when g(x) ≈ x, this isn't always true, so it's necessary find a possible generalization.
Generalizing the special case into one always applicable it's possible find a **full attack**, called **Backward Pass Differentiable**.
Definition:

- Let $f(\cdot)=f_{1...j}(\cdot)$
- $f_j(\cdot)$ is a non-differentiable (or not usefully-differentiable) layer.

Find $g(\cdot)$ such that $g(x)\approx f_i(x)$ (approximation), the is possible approximate $\nabla_x f(x)$ by performing the forward pass through $f(\cdot)$
Hence, in particular, computing a forward pass through $f_i(x)$,but not in the backward pass, where it's necessary replacing $g(x)=f_i(x)$,**this replacement is done only on the backward pass.**

"Applying BPDA often requires more iterations of gradient descent than without, because each individual gradient descent step is not exactly correct."

Applying BPDA on both the forward and backward pass isn't useful.

2.3 Stochastic

Stochastic gradients are find out when using **randomized transformations** to the input before feeding it to the classifier or in case of **stochastic classifier**,

Problem:
it is necessary to estimate the gradient of the stochastic function when is being used optimization-based attack.

Solution:
Hence, to correctly compute the gradient over the expected transformation to the input,. in defenses that employ randomized transformations to the input, is useful apply **Expectation over Transformation** (Athalye et al., 2017).
When attacking a classifier $f(\cdot)$ where before, input is randomly transforms with a function $t(\cdot)$ from a distribution of transformations T, EOT is used to optimizes the expectation over the transformation $E_{tsimT}\, f(t(x))$
This optimization problem can be solved by gradient descent, noting that $\nabla E_{tsimT}\, f(t(x))=E_{tsimT}\nabla f(t(x))$,**so differentiating through the classifier and transformation, and approximating the expectation with samples at each gradient descent step.**

2.4 Vanishing

Reparametrization is the main way this allow attacker to solve vanishing/exploding gradient, and circumvent defenses.

Problem:
Given a classifier $f(g(x))$ where $g(\cdot)$ performs some optimization loop to transform the input x to a new input x', this optimization loop means that differentiating with $g(\cdot)$, can lead to exploding or vanishing gradients.

Solution:
Using a change-of-variable $x=h(z)$ some function $h(\cdot)$ such that $g(h(z))=h(z)$ for all z, where $h(\cdot)$ is differentiable. In e.g if $g(\cdot)$ projects samples to some manifold in a specific manner, we might construct $h(z)$ to return points exclusively on the manifold. With this approach is possible compute gradients through $f(h(z))$ and finally circumvent the defense.

Observation
At the end, it's possible find at least one way to overcome the defense that present these characteristic gradients, that are like cracks and fissures in the "defense wall"

[1][2][5]

3. Adversarial Example Generation Methods and Defense Strategy

Following defenses are considered designed for the white-box setting, hence the adversary has full access to the neural network classifier, in its architecture, weights, parameters and defense, in this paper is reported that ICLR 2018 defenses suffer for gradient masking, all of the evaluated defenses have been tested under threat model in which it claims to be secure. It often easy to find imperceptibly perturbed adversarial examples by violating the threat model, but by doing so under the original threat model, in is shown that the original evaluations were inadequate and the claims of defenses' security were incorrect. Obfuscated gradients,, in other words are phenomenons formed from defenses that makes standard gradient-based methods fail to generate adversarial examples, a generalization of these possible attacks techniques to bypass three different types of obfuscated gradients can be done, and is after described.

Defining attack methods
Adversarial examples are build with iterative optimization-based methods. For a given instance x, these attacks attempt to search for a δ such that $c(x+\delta)\neq c'(x)$ either minimizing $\|\delta\|$, or maximizing the classification loss on $f(x+\delta)$.
To generate l_{00} bounded adversarial examples in [1] use **Projected Gradient Descent (PGD)** confined to a specified l_{00} ball; for l_2 , they used the Lagrangian relaxation of Carlini & Wagner (2017c), generally are used from 100 to 10,000 iterations of gradient descent, that are needed to obtain convergence.

The specific choice of optimizer is far less important than choosing to use iterative optimization-based methods.

Observation
An observation can be done at this point, after, in this paper are described some possible techniques to overcome defenses, but when it's possible consider a defense violated, and when an a defense it's consider robust. Obviously there is a change of paradigm between attacker and defender, an **asymmetric vision**, while an attack is considered correctly done if overcome the defense at least in one case and the defense is violated, while a defense can be considered robust if and only if doesn't exit any attack that has ever violated it.

Possible attack techniques to overcome defenses
Generating adversarial examples through optimization-based methods requires useful gradients obtained through backpropagation. Many defenses therefore either intentionally or unintentionally cause gradient descent to fail because of obfuscated gradients caused by gradient shattering, stochastic gradients, or vanishing/exploding gradients. We discuss a number of techniques that in they develop to overcome obfuscated gradients in [1].

Targeted vs. Non-Targeted: In some of generation methods, output of the classifier for the adversarial example can be specified (Targeted generation methods), while in others, can be arbitrarily chosen in the optimization (Non-targeted generation methods).

3.1 Methods

Here are described the most important and used methods to generate adversarial examples, each of them has a short and simply explanation.

One-Pixel Attack, this kind of attack that generate adversarial images by perturbing only one dimension of the input data, not perceptible to human eyes. One-Pixel attack solves following optimization problem:

maximize : $f_{target}(x_i)$

where : $\|x_i - x_i'\|_0 \leq d$

$f_{target}: \mathbb{R}^d \rightarrow [0,1]$ Is the probability of input belonging target class(wrong class) , while d is the number of pixels to be perturbed for generating adversarial example and in this particular case of one-pixel attack, d = 1. Differential Evolution method is a evolutionary algorithm used to solve the optimization problem, **one-pixel attack only requires class probabilities assigned by classifier for given instance, it's not required know the gradient.**

ZOO (Zeroth Order Optimization) Zeroth order optimization method to solve the following optimization problem, is used zeroth order gradient descent optimization method:

minimize : $\|x_i - x_i'\|_2^2 + c \cdot g_{tagert}(x_i')$

where : $x_i' \in [0,1]^d$

Zeroth order gradient descent optimization method does not require gradient of the objective function, it can be classified as black-box method. Derivative respect to all axis directions is calculate.

L-BFGS, is an optimization algorithm, here a function is minimized over constraint value using an inverse Hessian matrix approximated., the adversarial example is constructed inside the unit hypercube c is a suitable constant that regularizes the norm of the adversarial perturbation while the equation is minimizing the error for perturbed example.

minimize: $\|\Delta_x\|_2 + J(\theta, x + \Delta_x, y')$

where: $x + \Delta_x \in [0,1]^d$

Hot/Cold, used to find multiple adversarial instances for each input image, with the following optimization problem:

minimize: $1 - PASS(x_i - x_1')$

where: $f(x_i') = target$
$PASS(x_i, x_i') \geq \gamma$

Psychometric Perceptual Adversarial Similarity Score (PASS) measure the distance between adversarial example and given image as an alternative to L metrics, **PASS** measures the similarity perceived by human perception, **this permit small rotation and translation as long as they are imperceptible by humans**
Hot/Cold Method iteratively optimizes the above optimization problem by defining original class of the input as Cold and the target class as Hot, and moving from Cold to Hot.

FGSM (Fast Gradient Sign Method) It is a one-shot algorithm that is fast and consists of just one gradient update. The adversarial perturbation is one step towards the gradient in the input space, the sign of the gradient vector is scaled by ε and added to the original example to get the adversarial example.

$$\Delta_x = \varepsilon sign(\nabla J_\theta(x, l))$$

Natural-GAN : Generative Adversarial Networks (GANs) are models that consists of two models: a generative model and a discriminative model, the fist model generates adversarial examples to fool the discriminative model, whereas the discriminative model tries to capture whether the data is from the actual data distribution or generated by the generative model exist different versions
of GANs.

3.2 Reactive Strategy vs Proactive Strategy

There are many application that can be completely unusable if an attacker can circumvent the neural network, for example recognition of texts, faces, car plates, and the entire world of autonomous driving is under threat, is very easy imagine what kind of dangerous situations can happen for humans safety. It's already been introduced why is so important find a solution against adversarial examples, so after this short introduction we can do an important classification between defense strategies. Hence exist two main classes, **Reactive** and **Proactive** defenses. In order to generate an adversarial example is necessary have access to gradient in the network, let's do some history.

A possible approach that has been found not very useful is hide the probabilities vectors, these can be used to adjust the perturbation, finding when a single small change improve the probability that an image belong to a particular wrong label. An approach is hide the probabilities vector and show only final result, the final label.

However this defense is not particular effective, because it's possible find a “label border”, where

the final result label is switched from one to another, this can be used to find a correct perturbation that improve the probability of a specific class, this can be done using for example Computer Vision API of Google, Microsoft and Amazon platforms. Hence all defenses that implements their defense only hiding their gradient (Black Box Attack) can be circumvent.

In this **reactive** strategy there is a second network that is simply used to detect an adversarial example, so the defense is activate whatever an AE is find , this is going to reject the example. This approach can be very useful but there are many handicap, because to implement a system like this is necessary double the entire infrastructure.

The second strategy is called **proactive**, in few words this approach expect that AE are used inside the training loop of your neural network. Adversarial Training is largely used, and is very effective in improving performance.

[1][5][6]

4. Evaluating Methods

and

Results on ICLR 2018 Defenses

Many approaches can be used in evaluating robustness of defenses, it is not restrict to the computational power of an adversary artificially,hence if two defenses are equally robust but generating adversarial examples ,in fist case,takes one second and in the second takes ten seconds, the robustness has not increased, to do a correct evaluation it's also needed have specific thread models and testable claims.

Specific thread models and testable claims
So in a threat model precisely specified, the assumption that are made on this, are more robust.
For example, a possible complete description can be: " 85% of accuracy when
bounded by l_{00} distortion with $\varepsilon = 0.031$, with a full white-box access.

"A defense being specified completely, with all hyperparameters given, is a prerequisite for claims to be testable. Releasing source code and a pre-trained model along with the paper describing a specific threat model and robustness claims is perhaps the most useful method of making testable claims."

Against adaptive attacks
A defense, to be robust must be effective not only against existing attacks, but also against future attacks in a specified threat model, so attempt an adaptive attack is a necessary component of any defense proposal Main attack approach is to perform many attacks and report the mean over the best attack per image, this is done for a set of attacks $a \in A$ instead of reporting the value min, an adaptive attack is one that considers knowledge of the new defense.
Adversarial examples can be stopped by defenses that are designed to defending against not only existing attacks but also future attacks that may be developed.

As indicates in [1] a possible case of study are the ICLR 2018 non-certified defenses that argue robustness in a white-box threat model. After a short introduction is reported what they have found on these nine defenses.
They found that seven of these nine defenses present obfuscated gradient, while only two grant what are claiming.

[1]

4.1 Non-Obfuscated Gradient

4.1.1 Adversarial Training

Is proposed by *Goodfellow et al(2014)*, adversarial training solve a min-max game using a simple training on adversarial examples until the model learns to classify them correctly, it's a proactive defense.
Given training data X and a loss function l(·), standard training chooses network weights θ as

$$\theta' = arg\,min_\theta E\, l(x; y; F_\theta)$$

There is another important training approach that is used to train model on adversarial examples and is proposed by *Madry et al. (2018)*

which for a given ε ball solves

$$\theta' = arg\,min_\theta E[max\, l(x+\delta; y; F_\theta)]$$

They use a projected gradient descent.(PGD) to solve the inner maximization problem by generating adversarial examples, this kind of defense don't cause Obfuscated Gradient.
There are same limit, in fact adversarial retraining has been shown to be difficult at ImageNet scale and training exclusively on l_{00} ,adversarial examples grant a limited robustness to adversarial examples under other distortion metrics.

4.1.2Cascade Adversarial Training

Obfuscated Gradients are avoided also by Cascade adversarial trainig (Na et al., 2018) the main difference with the previous one is that instead of using iterative methods to generate adversarial examples at each mini-batch, is trained a first model, after are generated adversarial examples with iterative methods, on that model, added these to the training set, and then trained a second model on the augmented dataset.

4.2 Gradient Shattering

4.2.1 Thermometer Encoding

Neural networks behave in a largely linear manner and thermometer encoding try to break the linear manner.

Given an image x, for each pixel color x_i,j,c, the l-level thermometer encoding $\tau(x_i, j, c)$ is a l-dimensional vector where $\tau(x_i, j, c)k = 1$ if $\tau(x_i, j, c) > k/l$, and 0 otherwise (e.g., for a 10-level thermometer encoding, τ (0.66) = 1111110000).
It is not possible to directly use gradient descent on a thermometer encoded neural network because the discrete nature of thermometer encoded values,it's necessary build a Logit-Space Projected Gradient Ascent (LS-PGA) as an attack over the discrete thermometer encoded inputs.

Using this attack, the authors[1] perform the adversarial training of Madry et al. (2018) on thermometer encoded networks.
The intention of this defense is to break the local linearity of neural networks, in [1] is observed that model accuracy is weaker than the original Madry et al. (2018) model that does not use thermometer encoding. Because this model is trained against the LS-PGA attack, it is unable to adapt to the stronger attack that they present in[1].

4.2.2 Input Transformation

Guo et al. (2018) argue another type of defense, that propose a five input transformations to counter adversarial examples. In fist images are performed with cropping and rescaling, bit-depth reduction, and JPEG compression.
In a second moment there are two other kind of transformations :

- (a) randomly drop pixels and restore them by performing total variance minimization.
- (b) image quilting: reconstruct images by replacing small patches with patches from "clean"images, using minimum graph cuts in overlapping boundary regions to remove edge artifacts.

.
Using different combinations of input transformations along with different underlying ImageNet classifiers, including adversarially trained models, the author claim protection even with a vanilla classifier, but in [1] find that is possible bypass each defense independently from the combination. Circumvent image cropping and rescaling with a direct application of Expectation Over Transformation (EOT), bit-depth reduction and JPEG compression are circumvented with a BPDA and approximate the backward pass with the identity function for the total variance minimization and image quilting, which are both non-differentiable and randomized, it's also used EOT and use BPDA to approximate the gradient through the transformation.

4.2.3 Local Intrinsic Dimensionality (LID)

A general-purpose metric that measures the distance from an input to its neighbors, design by Ma et al. (2018) to characterize properties of adversarial examples.
This classifier is not intended as a defense against adversarial examples is a robust method for detecting adversarial examples.
In [1] they found that LID is not able to detect high confidence adversarial examples,Carlini & Wagner (2017),even in the unrealistic threat model where the adversary is entirely oblivious to the defense and generates adversarial examples on the original classifier.

4.3 Stochastic Gradients

4.3.1 Stochastic Activation Pruning (SAP)

SAP (Dhillon et al., 2018) use randomness into the evaluation of a neural network to defend against adversarial examples. SAP randomly drops some neurons of each layer f_i to 0 with a probability proportional to their absolute value, using dropout at each layer where instead of dropping with uniform probability, nodes are dropped with a weighted distribution. Values which are retained are scaled up to retain accuracy.
Applying SAP decreases classification accuracy slightly but improve the robustness,t he estimation of the gradients by computing the expectation over instantiations (EOI) of randomness.
At each iteration of gradient descent, instead of taking a step in the direction of $\nabla_x f(x)$ we move in the direction of $\sum_i \nabla_{xf}(x)$ from i=1 to K

4.3.2 Mitigating Through Randomization

Xie et al. (2018) defend against adversarial examples by adding a randomization layer before the input to the classifier. For a classifier that takes a 299 × 299 input, the defense first randomly rescales the image to a r × r image, with r ∈[299, 331), and then randomly zero-pads the image so that the result is 331×331, after the output is send to the classifier.
Considering three attack scenarios:

- vanilla attack (an attack on the original classifier),
- single-pattern attack (an attack assuming some fixed randomization pattern).
- ensemble-pattern attack (an attack over a small ensemble of fixed randomization patterns).

It's possible bypass this defense by applying EOT, optimizing over the distribution of transformations.

4.4 Vanishing/Exploding Gradient

4.4.1 PixelDefend

Song et al. (2018) propose using a PixelCNN generative model to project a potential adversarial example back onto the data manifold before feeding it into a classifier adversarial examples mainly lie in the low-probability region of the data distribution. PixelDefend is used to filter adversarially perturbed images prior to classification by using a decoding procedure to approximate finding the highest probability example within an ε-ball of the input image, reactive strategy.
In [1] they use PixelDefend on CIFAR-10 over various classifiers and perturbation values.
With a maximum l_{00} perturbation of ε = 0.031, authors dismiss the possibility of end-to-end attacks on PixelDefend due to the difficulty of differentiating through an unrolled version of PixelDefend due to vanishing gradients and computation cost.
Instead in [1] the problem of computing gradients is circumvent through an unrolled version of PixelDefend by approximating gradients with BPDA, and they mount an end-to-end attack using this technique. this attack,we can reduce the accuracy of a naturally trained classifier which achieves 95% accuracy to 9% with a maximum l_{00} perturbation of ε = 0.031. Hence combining adversarial training (Madry et al., 2018) with PixelDefend provides no additional robustness over just using the adversarially trained classifier.

4.4.2 Defense-GAN

Samangouei et al., (2018) uses a Generative
Adversarial Network (Goodfellow et al., 2014a) to project samples in the manifold of the generator before classifying them, this defense is similar to PixelDefend, but using a GAN instead of a PixelCNN, becouse Defense-GAN was not argued secure on CIFAR-10, in [1] they use MNIST data.
Adversarial examples exist on the manifold defined by the generator , and so it's possible construct an adversarial
example:

$x' = G(z)$

so that $x' \approx x$

but $c(x) \neq c(x')$

As such, a perfect projector would not modify this example, because it exists on the manifold described by the generator. Imperfect gradient descent based approach using Defense-GAN does not perfectly preserve points on the manifold but constructing a second attack using BPDA to evade Defense-GAN, it's possible grant only a 45% rate of success.

[1][2][5][6]

5. Implementation

All the work done by [1] is very useful to understand the entire Adversarial Examples world, with the possible strategies to overcome these obfuscated gradient. In the paper are reported the source code with a new implementation of original defenses,i found, with little differences, the same result of[1], they not reported the code of one of the two defenses that are claimed secure.
So, i decided to propose, one possible way of implementation of Adversarial Training, finding my inspiration in [7] where are proposed many solutions.

Requirements

```
import tensorflow as tf

from tensorflow.keras.datasets import mnist, cifar10
from tensorflow.keras import Sequential
from tensorflow.keras.callbacks import LambdaCallback
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Dense, Flatten, Activation

import numpy as np
import random


import matplotlib.pyplot as plt
```

I test my implementation in Colab, using Tensoflow2.0 and Keras, that must be installed, the choice of a Generator Algorithm for Adversarial Example fell on **Fast Gradient Sign Method**, or FGSM. Like reported in [2] this is a :

	Black-Box	**White-Box**
Targeted	One-Pixel Attack [13], ZOO [2].	L-BFGS [14], Hot/Cold* [10]
Non-Targeted	Natural-GAN [17]	FGSM* [6] , FGVM* [11], FGSM-Momentum* [3]

It is a one-shot algorithm that is fast and consists of just one gradient update. The adversarial perturbation is one step towards the gradient in the input space, the sign of the gradient vector is scaled by ε and added to the original example to get the adversarial example.[1]
FGSM, is White-box attack and Non-Targeted, can be formalized with:

$\Delta_x = \varepsilon sign(\nabla J_x(\theta, x, l))$ Hence, is necessary compute the gradient of the cost function respect to the input,output and weights of the model, solved by: $\nabla J_x(\theta, x, l)$, while this is the mathematical representation of the loss of the model $J(\theta, x, l)$, θ are the parameters(weights) of the model, x, is the input to the model, and y is the output of the model.

After is applied sign on each term in the gradient, reducing it to a matrix of -1s, 0s, and 1s, is needed a scale down operation, for the perturbations, by multiplying them by some small float, ϵ after is explained how choice this value..

These approach can work independently on cifar10, cifar100 or mnist, to compare the result it's necessary choice the same dataset, so cifar10 or mnist are preferable than cifar100.

It's also necessary **process our data** in fact the dataset is divided into batches to prevent running **out of memory**. The CIFAR-10 dataset consists of 5 batches, where each image was expressed in in a row vector of 3072 elements(pixels), now divided in heigth*width* channels, so 32*32*3. To do this is necessary use a function called **reshape ,** that transforms an array to a new shape without changing its data, using two divisions:

- divide the row vector into 3 pieces, where each piece means each color channel each of them of 1024 elements
- divide each of the 3 pieces in 32*32, height and width

This cannot be used in tensoflow and matplotlib, so it's necessary use the function **transpose**.

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
img_rows, img_cols, channels = 32, 32, 3
num_classes = 10

x_train = x_train / 255
x_test = x_test / 255

x_train = x_train.reshape((-1, img_rows, img_cols, channels))
x_test = x_test.reshape((-1, img_rows, img_cols, channels))

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

print("Data shapes", x_test.shape, y_test.shape, x_train.shape, y_train.shape)
```

After all this, we can finally define the model, or import it, in this case is presenteded a very simple sequential model, using :

```
model = create_model()
model.summary()
```

The model is relatively simple with three convolutional layers, obviously is possible construct more complex model with more computational complexity.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 11, 11, 32)        1568
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 4, 4, 128)         36992
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 2, 2, 256)         295168
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 1, 1, 256)         0
_________________________________________________________________
dropout (Dropout)            (None, 1, 1, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 30)                7710
_________________________________________________________________
dropout_1 (Dropout)          (None, 30)                0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                620
_________________________________________________________________
dropout_2 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210
=================================================================
Total params: 342,268
Trainable params: 342,268
Non-trainable params: 0
_________________________________________________________________
```

Training, (this code can be passed if you load a pre-trained model) with 30 epochs, and a batch of size 256.

```
model.fit(x_train, y_train,
          batch_size=256,
          epochs=30,
          validation_data=(x_test, y_test))
```

This my simple model achive 63,2% of accuracy.

```
print("Base accuracy on regular images:", model.evaluate(x=x_test, y=y_test, verbose=0))
```

```
Base accuracy on regular images: [0.0516434982419014, 0.6327000260353088]
```

Creation of AE with FGSM,

So now we define a function that receives an image and the correct label, fist of all it's necessary convert the image in a tensor, obviously using this tensoflow2 features:

```
image = tf.cast(image, tf.float32)
```

In tensoflow2 there is a particular object called GradientTape that can see what happens to a specific tensor .This represent $J(\theta, x, l)$, where J is the loss between input x and predicted y, hence we can calculate the gradient, and then appling sign with tensoflow2, the definition of our FGSM is completed.

```
def adversarial_pattern(image, label):
    image = tf.cast(image, tf.float32)

    with tf.GradientTape() as tape:
        tape.watch(image)
        prediction = model(image)
        loss = tf.keras.losses.MSE(label, prediction)

    gradient = tape.gradient(loss, image)

    signed_grad = tf.sign(gradient)

    return signed_grad
```

It's possible directly fit the model on the generator, or generate a bunch of images and train the model on those images, these are the steps:

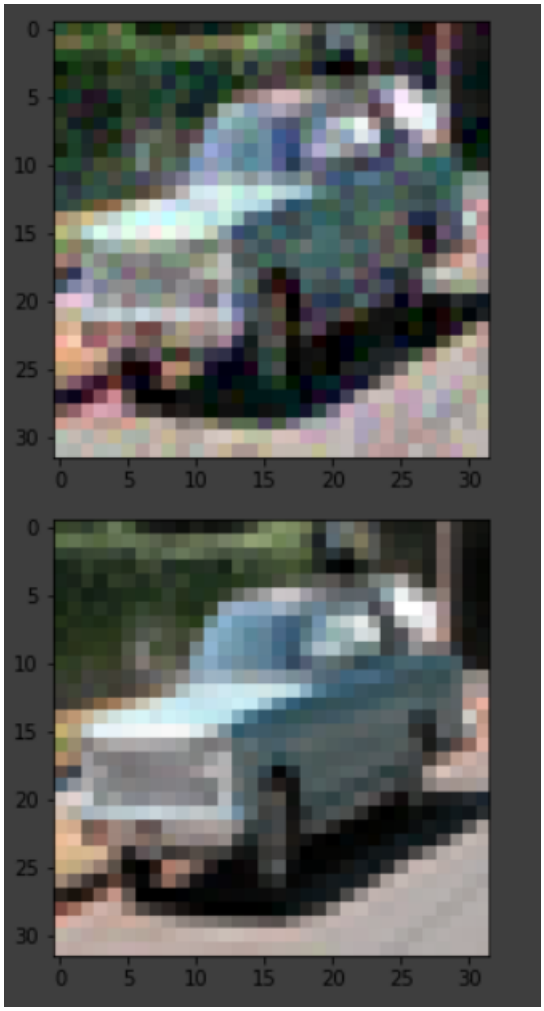
- In a loop, set up the x and y variables that will hold the adversarial images and their corresponding labels.
- pick the index of a random image
- can generate the perturbations
- add the perturbations
- append the adversarial image and its corresponding label to the array we have to hold them
- After we do that for every batch, we can finalize the x and y arrays, an yield them.

```
def generate_adversarials_find_EPSYLON(batch_size):
    while True:
        x = []
        y = []
        z= []
        for batch in range(batch_size):
            N = random.randint(0, 100)

            label = y_train[N]
            image = x_train[N]

            perturbations = adversarial_pattern(image.reshape((1, img_rows, img_cols, channels)), label).numpy()


            epsilon = 0.05
            adversarial = image + perturbations * epsilon



            x.append(adversarial)
            y.append(y_train[N])
            z.append(image)


        x = np.asarray(x).reshape((batch_size, img_rows, img_cols, channels))
        y = np.asarray(y)
        z = np.asarray(z).reshape((batch_size, img_rows, img_cols, channels))

        yield x, y, z



        adversarials, correct_labels, images = next(generate_adversarials_(4))
for adversarial, correct_label, image  in zip(adversarials, correct_labels, images):
    print('Prediction:', labels[model.predict(adversarial.reshape((1, img_rows, img_cols, channels))).argmax()], 'Truth:', labels[correct_label.argmax()])
    #show  perturbated images
    if channels == 1:
        plt.imshow(adversarial.reshape(img_rows, img_cols))
    else:
        plt.imshow(adversarial)
    plt.show()
    #show  original images
    if channels == 1:
        plt.imshow(image.reshape(img_rows, img_cols))
    else:
        plt.imshow(image)
    plt.show()
```

This code, can be extended with this version where a prepare version of generate_adversarial, is used to find a correct value for epsylon, not too much to make different images seem, comparing the original image, with the artificial.

```
def generate_adversarials(batch_size):
    while True:
        x = []
        y = []
        for batch in range(batch_size):
            N = random.randint(0, 100)

            label = y_train[N]
            image = x_train[N]

            perturbations = adversarial_pattern(image.reshape((1, img_rows, img_cols, channels)), label).numpy()


            epsilon = 0.1
            adversarial = image + perturbations * epsilon

            x.append(adversarial)
            y.append(y_train[N])


        x = np.asarray(x).reshape((batch_size, img_rows, img_cols, channels))
        y = np.asarray(y)

        yield x, y
```

I find that epsylon=0,1 was a too high value, so a new epsylon value is set to 0,05, therefore the car images below are now very similar.



This model can be evaluated with this FSGM (White-box attack), generating a set of 10,000 artificial images, my model has no defense, and therefore as expected his precision drops to 0%.

```
x_adversarial_train, y_adversarial_train = next(generate_adversarials(20000))
x_adversarial_test, y_adversarial_test = next(generate_adversarials(10000))

print("Base accuracy on adversarial images:", model.evaluate(x=x_adversarial_test, y=y_adversarial_test, verbose=0))
```

```
Base accuracy on adversarial images: [0.1827002763748169, 0.0]
```

Applying Adversial Training

There are two possible approaches now to use Adversarial Trainig, can either train the model directly on the generator or pre-generate a dataset of adversarial images to fit on. The benefits of a direct train method requires significantly more computing power, so it's reported the code to do the second approach, that is simpler and remains efficient. After we generate the training images, we can fit the model on them as we would to any other dataset.

```
model.fit(x_adversarial_train, y_adversarial_train,
          batch_size=256,
          epochs=30,
          validation_data=(x_test, y_test))
```

Testing my Defended Model

My model achieved an accuracy of 29.3% on these images(adversarial examples).
To increase this accuracy, we can alternate training the model on new adversarial images and generating new adversarial images. Fitting the model directly on the generator would also significantly improve its final accuracy.

[3][4][7]

References

[1] Anish Athalye, Nicholas Carlini , David Wagner ; Obfuscated Gradients Give a False Sense of Security Circumventing Defenses to Adversarial Examples; 31/07/2018; https://arxiv.org/abs/1802.00420

[2] Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy ; EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES ; 20/03/2015; https://arxiv.org/pdf/1412.6572.pdf

[3] Sebastian Theiler; Implementing Adversarial Attacks and Defenses in Keras & Tensorflow 2.0 ; 23/09/2019; https://medium.com/analytics-vidhya/implementing-adversarial-attacks-and-defenses-in-keras-tensorflow-2-0-cab6120c5715

[4] Park Chansung;CIFAR-10 Image Classification in TensorFlow ; 17/04/2018; https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c

[5]Xiaoyong Yuan, Pan He, Qile Zhu, Xiaolin Li ;Adversarial Examples: Attacks and Defenses for Deep Learning ; 07/07/2018; https://arxiv.org/pdf/1712.07107.pdf

[6] Ian Goodfellow, Nicolas Papernot, Sandy Huang, Rocky Duan, Pieter Abbeel, Jack Clark ; Attacking Machine Learning with Adversarial Examples; 24/02/2017; https://openai.com/blog/adversarial-example-research/

[7]Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu;CIFAR10 Adversarial Examples Challenge; https://github.com/MadryLab/cifar10_challenge