Digital Forensics Individual Project: One-pixel attack

Mirco Beltrame

1 Overview

Adversarial Machine Learning is the field of ML that studies possible attacks to ML algorithm. In the case of image classification the aim of an attack is to fool the model to predict a wrong label, by adding a perturbation p to the imput image I, modifying it. An attack is more successful if the perturbation is small enough to be imperceptible also to human eye.
But when is a perturbation "small"? When dealing with ML and DL models, images are represented as vectors and so are perturbations, so we can use different norms to identify different meanings of small.
In this project we will deal with One-pixel attacks, that means fooling a deep network by changing the colour of only one pixel in the image, this allows us to interpret it using the L_0 norm, that counts how many components different from zero a vector has.
We can have two different types of attacks: untargeted and targeted, the difference is that in untargeted attacks we just care that the model makes a wrong prediction, while in targeted ones we aim at a specific class the model will predict.
The result of an image classification model is a vector of probabilities, one for each class, that we can interpret as the confidence the model has that the image we showed it belongs to that class.
Let $\mathcal{I}$ the set of images and $\mathcal{L} = \{l_1, l_2, \ldots, l_n\}$ the set of the possible classes. Then the model we are attacking can be represented by $f : \mathcal{I} \longrightarrow \mathcal{L}$.
Finally let $I \in \mathcal{I}$ the image we are presenting, with true label $l \in \mathcal{L}$, then the probability of I belonging to the class $t \in \mathcal{L}$ is given by $f_t(I)$.
With this formalism the problem of adversarial attacks can be formulated as an optimization one:

- for **untargeted** we want to minimize the confidence of the model in the true label class
- for **targeted** we want to maximize the confidence of the model in the desired class

And in the case of the one-pixel attack, we have the restriction that

$$L_0(p) = 1$$

So formally we are solving, for the **untargeted** attack:

$$\arg\min_p \quad f_l(I + p) \qquad \text{subject to } L_0(p) = 1$$

and for the **targeted**:

$$\arg\max_p \quad f_t(I + p) \qquad \text{subject to } L_0(p) = 1 \quad t \neq l$$

that can be reformulated as

$$\arg\min_p \quad 1 - f_t(I + p) \qquad \text{subject to } L_0(p) = 1 \quad t \neq l$$

2 Differential evolution

To make the implementation easier, we can think of a perturbation as a 5-uple (x, y, r, g, b) and the image as are usually represented as a tensor of dimension $(w, h, 3)$, so we can think of it as three matrices, one for each color channel.
To perturb the image I with the perturbation p we will do the following:

$$I[x][y][0] = r$$

$$I[x][y][1] = g$$

$$I[x][y][2] = b$$

using an array-like notation. This is easier to implement because we don't have to worry about pixel values overflowing, the perturbation in the "additive" notation will be this perturbation minus the image's pixel value.
The problem we formulated above, using this notation as one main difficulty: the values x, y, r, g, b are integers and this makes the function to minimize a non continuous one and don't allow us to use standard gradient-based methods for minimization.
To solve this issue the authors of "One pixel attack for fooling deep networks" suggest using **Differential Evolution** (DE).
DE is a population based optimization algorithm that can be used whenever the gradient of the function to minimize is not computable (like in the case of a black box attack), or a gradient-based method gets stuck in local minima.
It is very useful for this task also because it requires low information given only by the output of the model.
DE works by generating candidate solutions starting from current candidates in a stochastic way:

$$p_i(g + 1) = p_{r_1}(g) + F(p_{r_2}(g) - p_{r_3}(g))$$

where p_i is a possible solution, $r1, r2, r3$ are different random numbers, F is a scale parameter and g is the index of generation.
By generating this candidates and keeping the ones that lower the value of the function, the algorithm arrives to a minimum.
In my code I relied on `scipy` implementation.

3 Implementation

For my implementation I used Python with the modules Numpy, Tensorflow and Scipy, and ran the code in Google Colab.

Dataset

As a dataset I used Cifar10, that contains 60000 colored images of size 32×32 pixels, divided in 10 different classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. As we can see we can cluster these classes in two macro goups: vehicles and animals.
I chose this dataset because is one one of the most used ones and the images are of a size such that the notebook could run in Colab for a reasonable amount of time.
However this attack doesn't exploit any specific characteristic of the dataset, making it functional for other datasets with little to no change in the implementation, but the effectivness of the attack could change.

Model

As a target model I used a pretrained ResNet50, trained on ImageNet, fine-tuned to Cifar10.
I obtained it thanks to Tensorflow, what I did is adding a last dense layer with 512 selu units (I chose this activation function after trying also relu and sigmoid) and an output layer of ten softmax units to obtain the class predictions.
Training this model for 5 epochs with the 50000 images trainig dataset allows it to obtain an accuracy of around 66% on the 10000 images in the test set.
This allows me to have a pool of about 6600 possible images to attack, because the attacks makes sense if the model c orrectly classify the image in the first place.

Main functions

The attack is composed of three main functions:

- `attack_success(model, image, perturbation, true_label, target = None)`
- `function_to_minimize(model, image, perturbation, true_label, target = None)`
- `attack(model, image, true_label, target=None)`

attack_success

This function's role is to evaluate whether a perturbation is such that the attack is successful.
It works by perturbing the image and making the model predict it. Then if we are doing an untargeted attack it checks if the predicted label is different from the true one, if we are doing a targeted one it checks if the predicted label is equal to the desired one.
If the attack succeed it will return `True` otherwise it will return `None`, this is due to Scipy requirements on the callback function. Its main purpose is to serve as a callback function to early stop the DE algorithm when an acceptable perturbation is found.

function_to_minimize

This is our objective function, what it does is computing the prediction of the model for the perturbed image and returning the true label confidence in untargeted attacks and 1 minus the desired label confidence in targeted ones.

attack

This is where the magic happens!
In this function we call scipy's `differential_evolution` function, it requires as the main parameter the function to minimize `func`. Some additional parameters are very useful such as the region where to search the solution `bounds` and the callback function `callback`.
To make the function usable by `differential_evolution` we have to fix the parameters of our `function_to_minimize` (model, image, true label and target label) and make it dependend only of the perturbation, to do so we wrap it inside a function called `function`.
The same is done for the `callback` function that simply calls `attack_success` with the fixed parameters.
As bounds we use $[0, 32] \times [0, 32] \times [-128, 128] \times [-128, 128] \times [-128, 128]$ because the images are 32×32 and the Resnet50 model wants BGR values in that interval.

Other functions

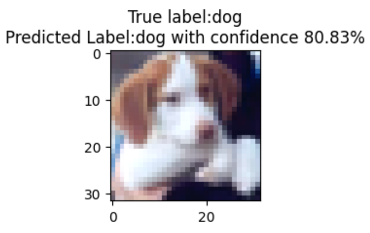
Other important functions to consider are:

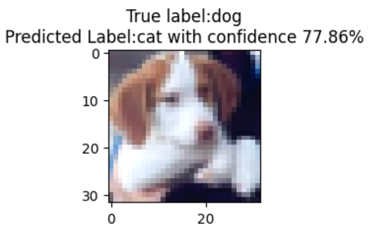
- `perturb_image` that outputs the perturbed image given a perturbation
- `corrects` that outputs the list of all the images the model correctly classify
- `plot_image` that plots the image with true label, predicted label and relative confidence

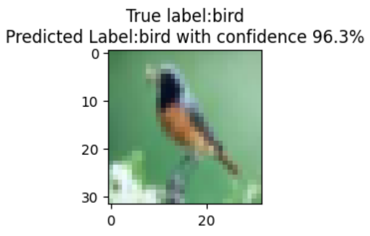
4 Results

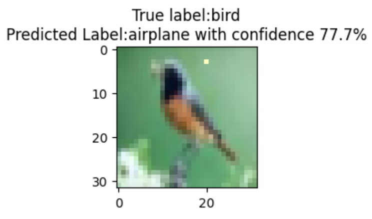
Untargeted attacks

Succeding in an untargeted attack is normally easier to achieve, intuitively we have more classes that we value as correct while in targeted only one is valid.
Testing the attack in 300 images correctly classified by the model I obtained 215 successful attacks, so a success rate of about 71%.
What is interesting to notice is that running the experiment in Google Colab took approximately 4 and a half hours, so a pretty long time. But how was this time split between the successful attack and the unsuccessful ones?
Successful attacks took cumulatively 9084 seconds, so an average of 42 seconds per image. That means to attack the remaining 85 images the attack took approximately 2 hours, making it an average of 84 seconds per image, without succeding.
This certainly has to do with the `maxiter` parameter of DE, that gets reached when the attack doesn't succeed, while stays pretty low when the attack manage to work.
We have to notice that this result highly depends on the strength of the model and the dimension of the images, so the `maxiter` hyperparameter has to be tuned to the model and dataset.
Here are some examples of successful attacks.





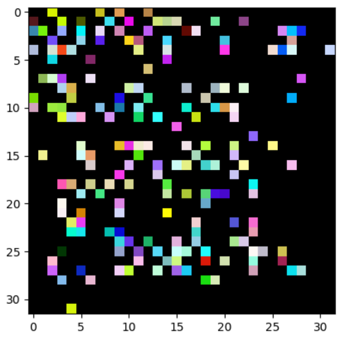




Notice how in the dog image the perturbation is almost not noticeable, especially if you don't have the true image to compare, while the bird one is pretty easily detectable by human eye.

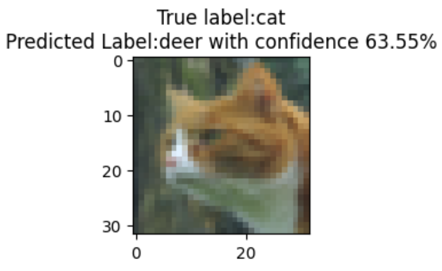
Perturbation distribution

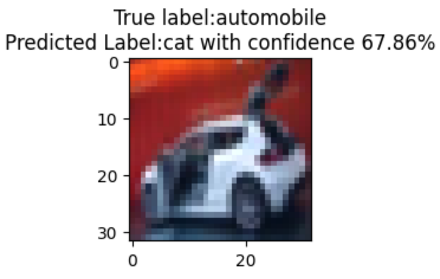
As we can see, the perturbation are more frequent in the center of the image rather than in the edges and this could lead to think that perturbing where the subjects is located is a better choice to have an attack succeed, but is more noticeable from the human eye.
As for the color, those are pretty random.

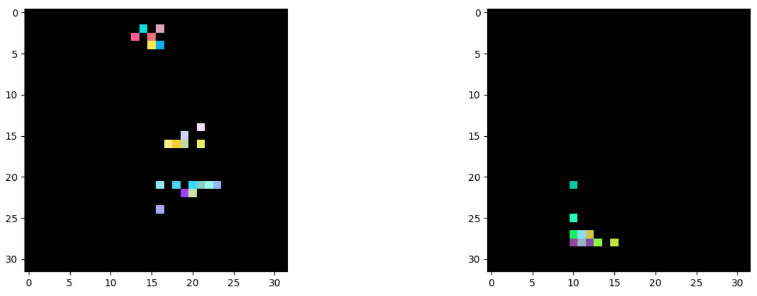


Same-image attack

One interesting experiment is to see how the perturbations behave in a single image, do they spread around? Or do they tend to stay in a specific zone? Do they change color a lot?
The answers seems to be that some zones of the image are more sensible than others and tend to misguide the model to the same class.
Trying 50 attacks on a cat image and an automobile image resulted in being able to point out this regions.
In the cat image (left) modifying the top region and the middle region results in a dog classification, and the lower part in a deer classification.
In the car image (right) we can see that perturbations cluster in a single zone and the modified images result in cats.





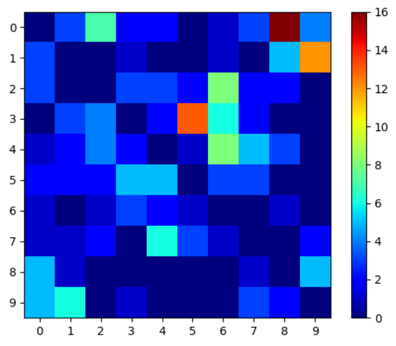


Confusion matrix

In the 300 sample I chose, the distribution of the classes is not uniform, but there is not a very big inbalance.
In the table we can see the distribution of classes in the 215 successful attacks

class	number of samples
0 - airplane	21
1 - automobile	18
2 - bird	20
3 - cat	17
4 - deer	20
5 - dog	20
6 - frog	28
7 - horse	19
8 - ship	29
9 - truck	23

And here we can see a confusion matrix with the true labels on the vertical and the predicted on horizontal.



As we can see the images belonging to vehicle classes (airplane, automobile, ship, truck) usually stay in the same cluster of vehicles, suggesting that internally, the model sees and represent

these images close toghether and moving to one of these class to the other is easier.
One other interesting fact we can notice is that usually a cat is modified into a dog, but the relation is not reciprocal, as if the cat cluster is near the dog one, but further apart from other animals.
Finally we can see that birds and deers get sent to frogs more frequently than other. Also, as one could expect airplanes can easily be modified also into birds.

Targeted attack

I experimented targeted attacks by using as a target 9 - l with l the true label, this is an absolutely arbitrary choice to be sure to obtain a target that is not the true label.
Targeted attacks have a high failure rate, trying 100 of them with this characteristic I obtained only 26 of success, a much lower rate than untargeted.
It has to be noted that no particular choice in the target was made and this could have made the performance not very good.
A strategy to choose a more probable target to the attack could be to use the confusion matrix of the untargeted attack, and select the more common predicted classes.

5 Conclusions

With this project I learned that attacking a model is an useful tool also to understand better the model and how it works and represent images internally.
The next step could be trying to attack more complex models or different types of models, like a language model, to see if a small change in a phrase, like a single letter, that could be interpreted by an human user as a typing error, could have a big impact in the output of such a model.

6 Refereneces

The main resource for this project is the paper ”One pixel attack for fooling deep networks” by Jiawei Su, Danilo Vasconcellos Vargas, Sakurai Kouichi.
For some implementation details I relied on a GitHub repository, implementing the attack, by Dan Kondratyuk that can be found here