Adversarial attacks and adversarial training
Master Project

Authors:

Amin El Iraki
Olivier Randavel
Hadrien Mariaccia
Kenza Hammou

May 3, 2020

Contents

1 Introduction

This project aims to test and compare several adversarial attacks against a standard CNN model trained on CIFAR10 dataset [3] and to perform adversarial training to improve model robustness. An adversarial attacks is a specific noise added to a picture in order to fool the model. The human is seeing the same picture but the machine is predicted another label.

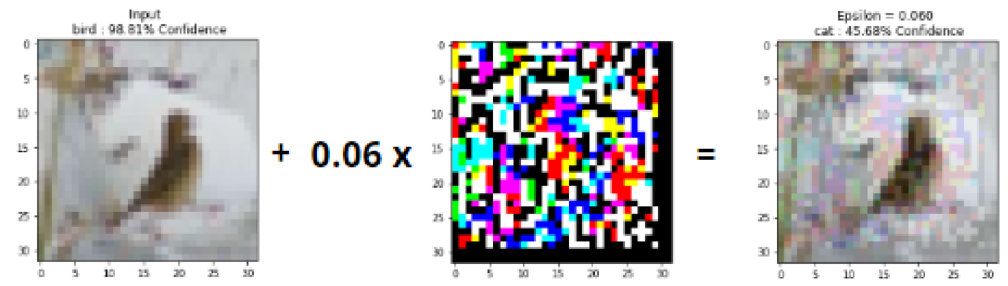
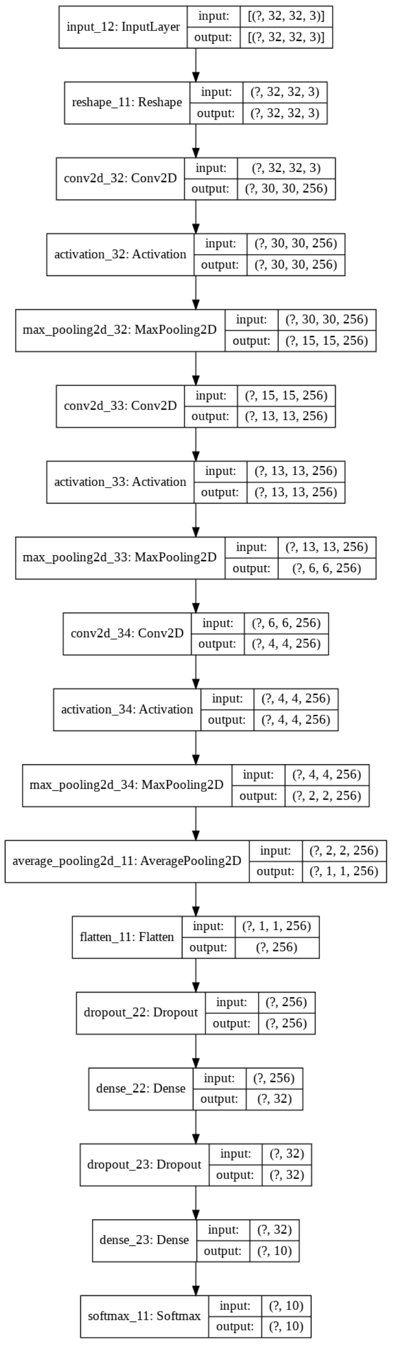


Figure 1: Showing the effect of the FGSM-attack on the CIFAR10 dataset

In this project, we are using the *Keras.tensorflow* library. We could also have used *Pytorch*. The data used is CIFAR10 and it is availaible under the library *keras.datasets*. All our implementations can be found in this github repository [1]. The project is divided in three parts. First we will present our model used through the project. Then, the second point will emphasize on fooling the model with the l_∞-PGD attack and perform adversarial training. Finally, several adversarial attacks will be evaluated against our standard model.

2 Our model

2.1 Architecture


input_12: InputLayer | input: [(?, 32, 32, 3)] | output: [(?, 32, 32, 3)]
reshape_11: Reshape | input: (?, 32, 32, 3) | output: (?, 32, 32, 3)
conv2d_32: Conv2D | input: (?, 32, 32, 3) | output: (?, 30, 30, 256)
activation_32: Activation | input: (?, 30, 30, 256) | output: (?, 30, 30, 256)
max_pooling2d_32: MaxPooling2D | input: (?, 30, 30, 256) | output: (?, 15, 15, 256)
conv2d_33: Conv2D | input: (?, 15, 15, 256) | output: (?, 13, 13, 256)
activation_33: Activation | input: (?, 13, 13, 256) | output: (?, 13, 13, 256)
max_pooling2d_33: MaxPooling2D | input: (?, 13, 13, 256) | output: (?, 6, 6, 256)
conv2d_34: Conv2D | input: (?, 6, 6, 256) | output: (?, 4, 4, 256)
activation_34: Activation | input: (?, 4, 4, 256) | output: (?, 4, 4, 256)
max_pooling2d_34: MaxPooling2D | input: (?, 4, 4, 256) | output: (?, 2, 2, 256)
average_pooling2d_11: AveragePooling2D | input: (?, 2, 2, 256) | output: (?, 1, 1, 256)
flatten_11: Flatten | input: (?, 1, 1, 256) | output: (?, 256)
dropout_22: Dropout | input: (?, 256) | output: (?, 256)
dense_22: Dense | input: (?, 256) | output: (?, 32)
dropout_23: Dropout | input: (?, 32) | output: (?, 32)
dense_23: Dense | input: (?, 32) | output: (?, 10)
softmax_11: Softmax | input: (?, 10) | output: (?, 10)

2.2 Parameters

- epochs : 100
- batch size : 128
- learning rate : 0.01
- decay : 10^{-6}
- validation set : 10%
- Kernel size on each convolutional layer : 3

2.3 Performances

- accuracy : 78.5%

3 Performing an adversarial attack and an adversarial training

3.1 Coding l_∞-PGD attack and evaluating the robustness of the model

3.1.1 l_∞-PGD attack

In this part we have implemented from scratch the l_∞-PGD attack using this paper [5] as a reference. This method is iterative and is built on the First Gradient Sign Method (FGSM) algorithm ,which is a white box attack whose goal is to ensure misclassification. A white box attack is characterised by a complete access to the model. The attacker knows all the weights and the gradient of the method. The FGSM aims to create a new image using the sign of the gradient of the loss and maximising the loss for a particular input image x. A coefficient δ is added to make this perturbation very small and undetectable by a human eye. This attack is done on already trained model.

$$attackx = x + \delta * sign(\nabla_x J(\theta, x, y)$$

The l_∞-PGD attack, Projected Gradient Descent is an iterative method. The input image is projected *num-iter* times in order to build a robust attack and we make sure that the projection stay in the circle of radius ϵ. Besides, we choose δ to characterise the coefficient of the perturbation.

$$x_{t+1} = P_{B_\infty(x,\epsilon)}[x_t + \delta * sign(\nabla_x J(\theta, x, y))]$$

3.1.2 Robustness

The first attack was performed usning FGSM method on the test data-set using $\delta = 0.03$. As recall, we previously ended the training with a model's accuracy of 78.5%. This attack is quite powerful because the accuracy resulting is under 10%. The attack is obviously undetectable by human eyes as shown below.

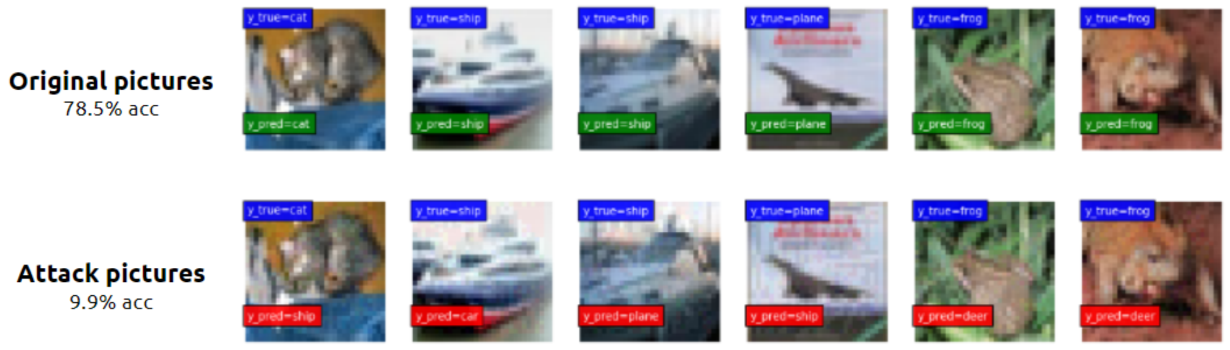


Figure 2: FGSM attack

The l_∞-PGD attack has been handled on the train data-set. We use specific parameters : $\delta = 0.003$, $\epsilon = 0.003$ and $num\_iter = 5$. The attack is very efficient and the accuracy fall under 4%. However the attack is quite detectable by human eye, but it is still possible to classify the pictures.

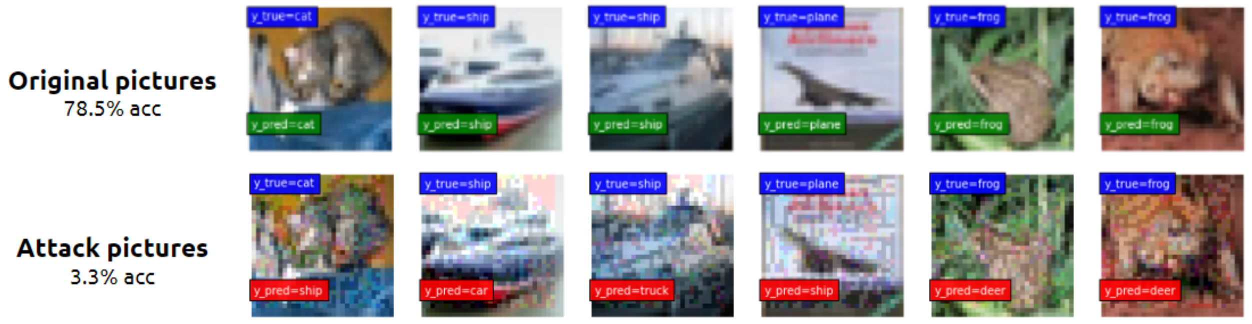


Figure 3: l_∞-PGD attack

3.2 Adversarial training

3.2.1 Model parameters

The parameters used for the adversarial training are the same than the standard training. Except the number of epochs, it is lower because to train with a generator, we had to choose a number of steps per epoch : the number of batch to train on within an epoch. With a standard training on CIFAR10 dataset, an epoch is completed after 352 batches of size 128. In order to train our robust model in a reasonable time we had to choose a lower number of steps per epoch : 20.

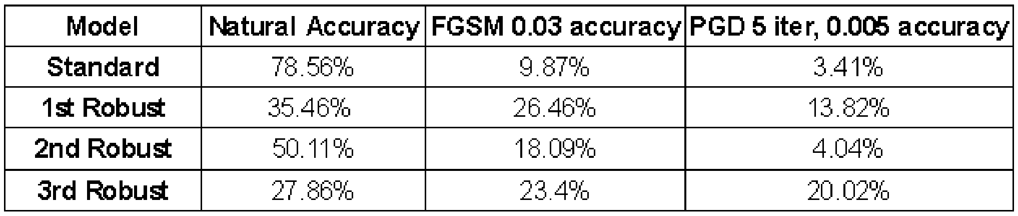
3.2.2 Attack generator to feed model at each batch

We fed our adversarial training model with attacked data transformed with FGSM attack with $\delta = 0.03$ and with PGD attack with $\epsilon = \delta = 0.005$ and 5 iterations. As Madry et al [5], we wanted to test both adversary to compare robustness of each.

1st generator : Generate only attacked data
To perform adversarial training, we implemented an attack generator. The idea is to attack each batch of data with the current model weights following the method proposed by Madry et al. [5]. In this manner the model will learn only on attacked data randomly chosen in the training data.

2nd generator : Generate attacked data and not attacked data
We observed by performing adversarial training only on attacked data with FGSM that the model learns to perform better on attacked data but is significantly worst in term of natural accuracy. Hence, we modified the generator in order to generate $p\%$ attacked data and $(100 - p)\%$ of not attacked data. In this manner, we force the model to generalize better.
One interesting notable behaviour of the model is that the lower p is, the less the model will learn to recognize attacked data. We observed that with $p < 80\%$, with a relatively short training time (2h on Google colab GPU), the model focus on not attacked data and stops learning to recognize attacked data.

3.2.3 Results



Model	Natural Accuracy	FGSM 0.03 accuracy	PGD 5 iter, 0.005 accuracy
Standard	78.56%	9.87%	3.41%
1st Robust	35.46%	26.46%	13.82%
2nd Robust	50.11%	18.09%	4.04%
3rd Robust	27.86%	23.4%	20.02%

Figure 4: Adversarial training results

- First robust : Adversarial training using FGSM adversary on the first generator.
- Second robust : Adversarial training using FGSM adversary on the second generator with $p = 80\%$.
- Third robust : Adversarial training using PGD adversary on the first generator with a kernel size of 5 on the first convolutional layer.

We can see that there's a trade off between natural accuracy and accuracy under attack. We discuss what could have been improved in the further improvements section.

4 Evaluate the most efficient attacks

This part aims to evaluate several adversarial attacks and to find the most efficient attack that keeps image readable by human eye. Adversarial attacks are available through the library named *Adversarial Robustness Toolbox* [6]. In this part we are performing adversarial attacks using :

- NewtonFool
- CarliniLInf
- Basic Iterative Method

All these methods will be explained in the next part. Besides, we are also using the attacks coded in the first part.

- FGSM
- l_∞-PGD

4.1 Adversarial attack methods

In this section we are comparing different adversarial attack methods. In order to compare our results we will used three different metrics:

- the model's prediction on attack test sample
- the perturbation : $\frac{1}{n}\sum_{i=1}^{n}|x_{i_{attacktest}} - x_{i_{test}}|$
- the human eye readability

4.1.1 NewtonFool

NewtonFool is detailed in this article [7]. It approximates the nearest classification boundary based on Newton's method for solving nonlinear equations.

Assumptions :

- the classifier$F(x)$ is of the form $argmax_l F_s(x)$
- the softmax output F_s is available to the attacker

Starting with x_0, they approximated $F(x_i)_l$ using a linear function step by step as follows :

$$F_s^l(x) \approx F_s^l(x_i) + \nabla F_s^l(x_i)(x - x_i) i \in [0, n] \rightarrow (1)$$

In equation (1) , x_0 is the original sample and $l = F(x_0)$. $d_i = x_{i+1} x_i$ is the perturbation at iteration i. They tried to find small d_i to decrease the value of the function F_s^l as fast as possible to 0 : $F(x_0+d_i)_l \approx 0$.

Different parameters were used in this method : η and max_iter (the maximum number of iterations). η characterizes the perturbation such that $\mid d_i \mid < \eta * \mid x_0 \mid$

4.1.2 CarliniL_∞

CarliniL_∞ is detailed in this article [2]. The aim is to misclassify x by adding a small noise to x. Carlini methods are part of the targeted attack. The approach is as followed :

$$\text{minimize } D(x, x+\delta) \text{ such that } C(x+\delta) = t, x+\delta \in [0,1]^n$$

The goal is to find the δ that changed the classification of x to t. But the result must be a valid image. The infinity characterised the L_∞ distance D. It measures the maximum change to any of the coordinates :

$$||x - x'||_\infty = max(|x_1 - x'_1|, .., |x_n - x'_n|)$$

The constraint $C(x+\delta) = t$ is not linear then it is difficult to optimize. Therefore the problem is reformulate as follow :

$$\text{f is the objective function such that } C(x+\delta) = t \text{ if and only if } f(x+\delta) \leq 0$$

The article presents several different objective function for f an $f_6(x') = (max_{i \neq t}(Z(x')_i - Z(x')_t)^+$ with Z the softmax function. Finally we are optimizing this problem with the stochastic gradient descent method :

$$\text{minimize } D(x, x+\delta) \text{ such that } f(x+\delta) \leq 0 \text{ , } x+\delta \in [0,1]^n$$

$$\text{minimize } D(x, x+\delta) + c * f(x+\delta) \text{ such that } x+\delta \in [0,1]^n \text{ with c a constant}$$

We set different parameters to this method, ϵ, max_iter and the $learning\_rate$. The ϵ value is used to define the limit of pixel modification. This helps to keep images readable by human eye. The paper also proves that the L_∞ is the most efficient distances between L_2 and L_0. The $learning\_rate$ is the coefficient of SGD optimizer and the max_iter parameter characterises the maximum number of times that the pixel will be modify to misclassify it.

4.1.3 Basic Iterative Method

Basic Iterative Method is detailed in this article [4]. It is an extension of the FGSM method. Goodfellow et al. 2014 proposed the fast gradient sign method FGSM as a simple way to generate adversarial examples:

$$X^{adv} = X + \epsilon sign(\nabla_X J(X, y_{true}))$$

This method is motivated by linearizing the cost function and solving for the perturbation that maximizes the cost subject to an L_∞ constraint. This method is simple and computationally efficient compared to more complex methods like LBFGS. In fact, it is referred as “fast” cause it doesn’t require an iterative procedure to compute adversarial examples, and thus is much faster than other considered methods. However it usually has a lower success rate.

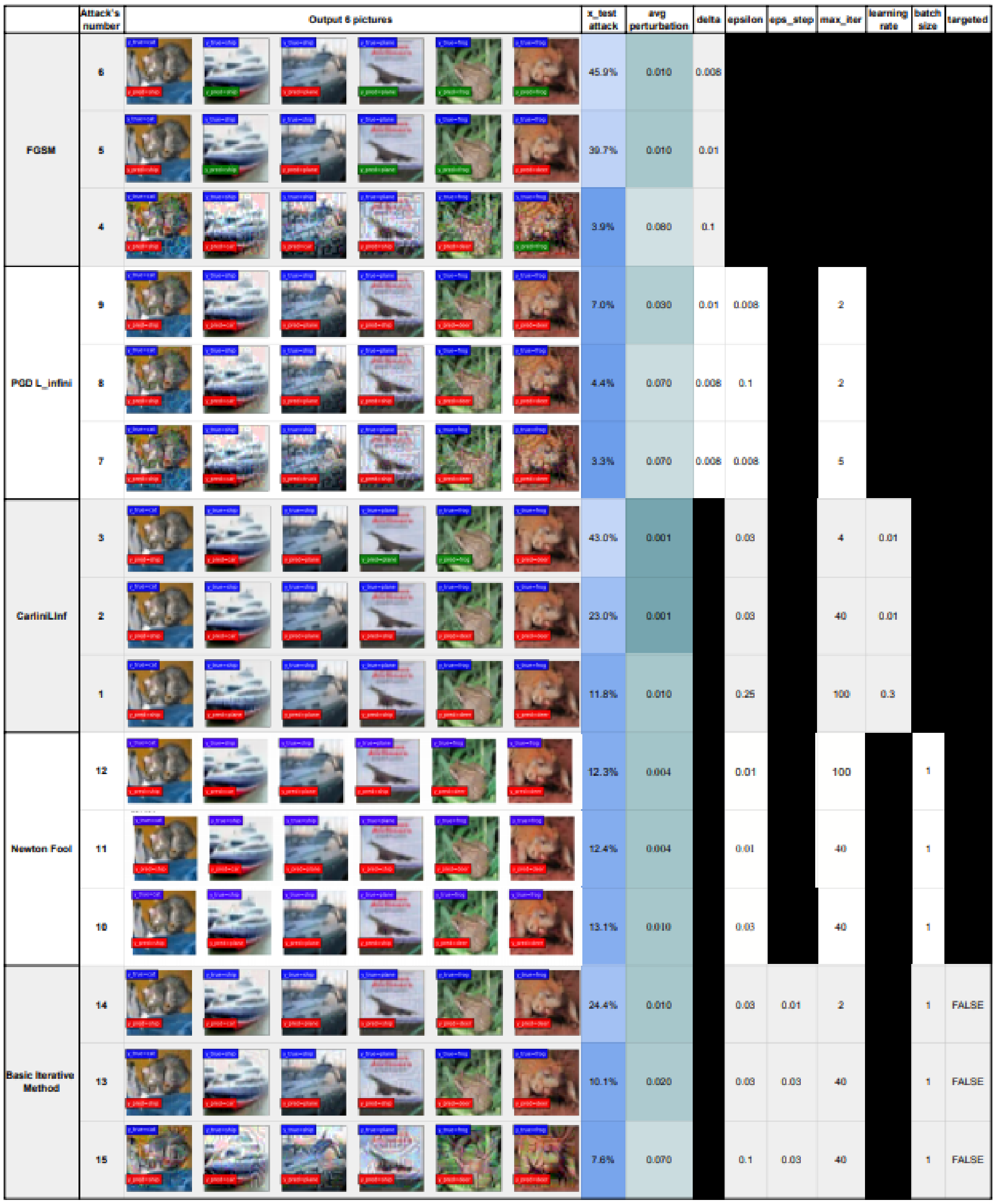
A way to extend this method is by applying it multiple times: that's the Basic Iterative Method. It consists of applying the FGSM method with a step size, and clip pixel values of intermediate results after each step to ensure that they are in an ϵ-neighbourhood of the original image:

$$X_0^{adv} = X, X_{N+1}^{adv} = Clip_{X,\epsilon}\left\{X_N^{adv} + \alpha sign(\nabla_{X_N^{adv}} J(X, y_{true}))\right\}$$

With $Clip_{X,\epsilon}(Z)$ being the function that clips the image Z element-wisely, such that each element $Z_{i,j}$ is in the range $[X_{i,j} - \epsilon, X_{i,j} + \epsilon]$.
The parameters for this method are:

- ϵ : the maximum perturbation that the attack can produce
- $maxiter$: the maximum number of iterations
- $targeted$: the indicator of whether or not the attack is targeted
- $batchsize$: the size of the batch on which the attack generate samples.

4.2 Evaluation



	Attack's number	Output 6 pictures	x_test attack	avg perturbation	delta	epsilon	eps_step	max_iter	learning rate	batch size	targeted
FGSM	6		45.9%	0.010	0.008						
	5		39.7%	0.010	0.01						
	4		3.9%	0.080	0.1						
PGD L_infini	9		7.0%	0.030	0.01	0.008		2			
	8		4.4%	0.070	0.008	0.1		2			
	7		3.3%	0.070	0.008	0.008		5			
CarliniLInf	3		43.0%	0.001		0.03		4	0.01		
	2		23.0%	0.001		0.03		40	0.01		
	1		11.8%	0.010		0.25		100	0.3		
Newton Fool	12		12.3%	0.004		0.01		100		1	
	11		12.4%	0.004		0.01		40		1	
	10		13.1%	0.010		0.03		40		1	
Basic Iterative Method	14		24.4%	0.010		0.03	0.01	2		1	FALSE
	13		10.1%	0.020		0.03	0.03	40		1	FALSE
	15		7.6%	0.070		0.1	0.03	40		1	FALSE

Figure 5: Attacks evaluation summary

In this part we made several experiments for each method and gathered the most interesting results. The two first methods (FGSM, PGD L_∞) give the best results with the lowest accuracy on the attack train set. We try different values of δ to increase the noise of the picture, however all pictures remain identifiable by human eye regarding the 6 pictures shown. But ones can argue that 6 pictures can't be representative of the whole sample. Following this remark, we will present in the next point a new visualization. Besides the three other methods of ART library have a bigger accuracy and these attacks seems to be less efficient. The perturbation is the same for our best PGDl_∞ method and the BIM method, but the accuracy is twice bigger.

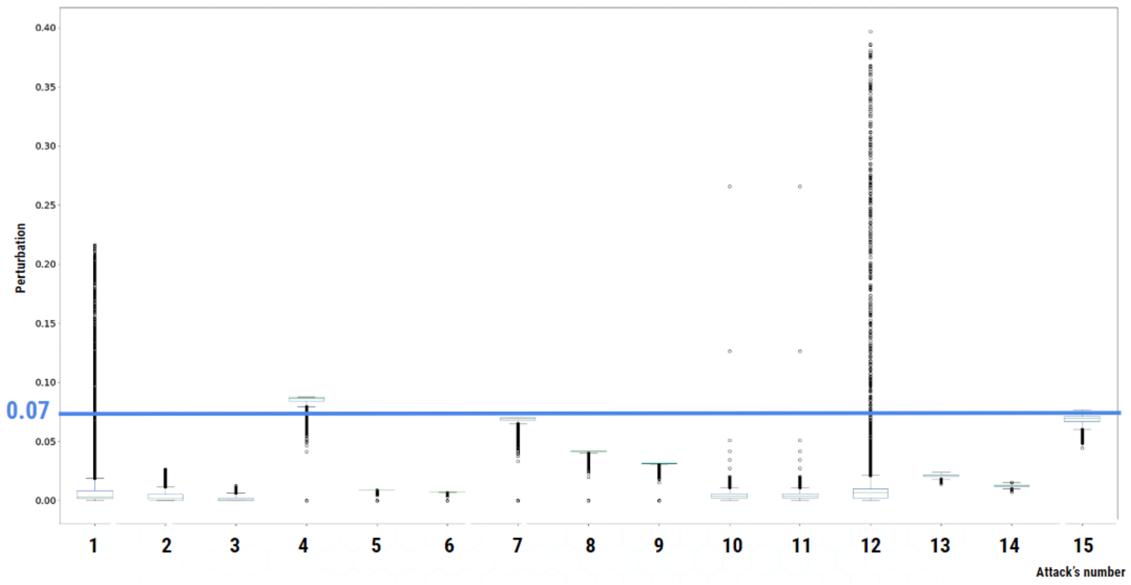


Figure 6: Box plots of image perturbation for each attack

This new visualization present the distribution of the perturbations among the whole sample of attack pictures. Empirically, we characterise a limit of perturbation where the human eye can't identify a picture. A perturbation that is bigger than 0.07 for the CIFAR10 data is too high. This dash helps to reject all attacks above the blue line. These boxplots show some weakness of our FGSM model (4), indeed several pictures are above the limit therefore this attack most be rejected. However our PGD L_∞ attack (7) remains under the line and has the lowest accuracy.

5 Further improvements

5.1 Adversarial training

5.1.1 Data Augmentation

In order to train a better data augmentation for adversarial training, we could have implemented the same generator as Madry et al [5]. This generator computes a uniform random perturbation on each image before attacking them with PGD attack. This perturbation stays in $B_\infty(x, \epsilon)$.

5.1.2 Model architecture

In order to increase model performance, we could have test other architectures with residual layers for example.

5.2 Resources

One big side of deep learning is to wait for models to train. During this project, more GPU resources would have been a non negligible advantage to test more things faster.

References

[1] Hadrien Mariaccia Kenza Hammou Amine El Iraki, Olivier Randavel. Adversarial_attack : Master project on adversarial attacks on cifar10 data. https://github.com/brash6/Adversarial_attack.

[2] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016.

[3] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[4] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.

[5] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks.

[6] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v1.2.0. *CoRR*, 1807.01069, 2018.

[7] Somesh Jha Uyeong Jang, Xi Wu. Objective metrics and gradient descent algorithms for adversarial examples in machine learning. 2017.