

Comment tromper les réseaux neuronaux ?

Adam Hotait
CentraleSupélec
Deep Learning, Computer Vision
`adam.hotait@student.ecp.fr`

Romain Pascual
CentraleSupélec
Computer Vision
`romain.pascual@student.ecp.fr`

Abstract

Nous avons étudié différentes méthodes pour tromper la classification d'un réseau neuronal. Après avoir montré que les réseaux neuronaux étaient robustes à des modifications aléatoires, nous avons mis en œuvre plusieurs méthodes en « boîte blanche » permettant de générer des adversarial examples : la méthode FGSM, la méthode FGSM itérative. Nous avons ensuite essayé de cibler quelle classe serait donnée par le réseau neuronal après modification des images.

1. Introduction

L'évolution récente des capacités du Deep Learning, reposant tant sur la disponibilité de données, l'amélioration des connaissances et la démocratisation des puissances de calculs, ouvre la voie à un monde où des algorithmes reposant sur des réseaux neuronaux prendraient des décisions complexes. Ainsi, des réseaux neuronaux pourraient, dans un futur proche, piloter des avions, conduire des voitures, prendre un diagnostic médical ou écrire des articles de presse.

Bien que de nombreux arguments existent aussi bien en faveur qu'en défaveur d'une telle utilisation de la technologie de Deep Learning, arguments qui dépassent pour la plupart le cadre de notre étude, nous avons choisi de nous intéresser à un argument majeur du camp des craintifs : le peu de fiabilité d'un réseau neuronal.

En effet, Szegedy et al. dans leur article *Intriguing properties of neural networks* [1] montrent que les réseaux de neurones sont vulnérables à des adversarial examples, c'est à dire des échantillons de données modifiées de manière imperceptible à l'œil humain, mais pouvant duper les réseaux neuronaux.

Goodfellow et. al., dans leur article *Explaining and Harnessing Adversarial Examples* [2] se sont intéressés plus en détail à ces adversarial examples et ont générés l'exemple bien connu en Figure 1.

Nous pouvons donc nous poser la question de la sécurité

des réseaux neuronaux : s'il est simple de modifier une image de manière imperceptible pour conduire à une mauvaise classification, ne serait-il pas possible pour une entité malicieuse de tromper un réseau neuronal, par exemple servant à reconnaissance des routes dans le cadre de la conduite autonome, pour prendre le contrôle du système ?

Afin d'étudier les risques liés aux *adversarial examples*, nous étudierons dans la suite différentes méthodes permettant de tromper un réseau neuronal. Nous commencerons par des méthodes de *misclassification* (dont le but est d'empêcher simplement la bonne classification) puis nous étudierons des méthodes de *target misclassification* (dont le but est de conduire à la classification vers une mauvaise classe choisie).

Notons qu'il existe deux types d'attaques : les attaques en boîte blanche, où nous connaissons l'architecture du réseau neuronal attaqué, et les attaques en boîte noire, où nous connaissons seulement les entrées et sortie du réseau neuronal attaqué. Bien que les attaques en boîte noire soient plus intéressantes car plus générales, nous allons procéder pour des raisons de simplicité à des attaques en boîte blanche.

2. Réseau neuronal et jeu de données

Nous utiliserons dans la suite de cette étude le réseau neuronal *ResNet18* [3], gagnant de l'ILSVRC (challenge de reconnaissance d'images ImageNet). Nous avons réduit le problème à 8 classes en utilisant le dataset LabelMe, qui contient 2600 images 256×256 réparties dans les catégories suivantes : autoroute, ville, côte, forêt, immeuble, rue, campagne et montagne [4].

La réduction à 8 classes s'est fait par *transfer learning* avec *fine-tuning* de *ResNet18* (batch size de 64, 100 epochs) en s'inspirant du tutoriel à ce sujet sur le site de PyTorch. Nous atteignons une *accuracy* de 0.964815 sur l'ensemble de validation.

Le code correspondant est disponible dans le fichier `transfer_learning.py`.

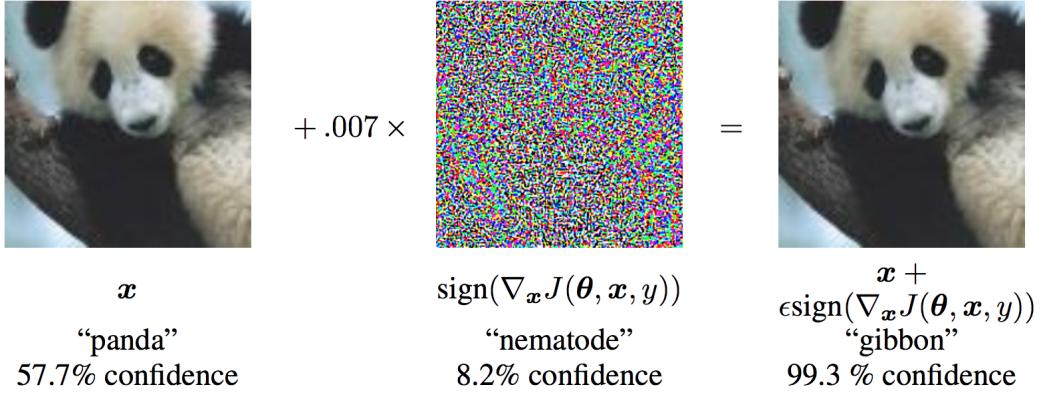


FIGURE 1. *Adversarial example* appliquée sur GoogLeNet. À gauche l'image originale classifiée comme un panda, au milieu la perturbation ajoutée avec un poids de 0.007 (invisible à l'œil humain), à droite l'image modifiée classifiée comme un gibbon.

3. Epsilon

Le but de cette attaque est de perturber le réseau neuronal en modifiant l'image sans être repéré par l'œil humain. Dans la figure 2, nous pouvons voir une image à laquelle nous avons ajouté un masque à plusieurs valeurs différentes de ϵ . Pour des $\epsilon < 0.05$, la perturbation est presque invisible, et elle devient très flagrante pour des $\epsilon > 0.3$.

Le code correspondant est disponible dans le fichier `epsilon.py`.

4. Attaque par bruit aléatoire

Nous pourrions penser en voyant le bruit ajouté de la Figure 1 qu'une simple dégradation aléatoire permettrait de dégrader suffisamment l'image pour tromper le réseau neuronal.

Nous avons mis en œuvre l'ajout d'une perturbation aléatoire de nos images avec des ϵ dans entre 0 et 0.5 en nous inspirant du tutoriel PyTorch sur la génération d'*adversarial examples* et le dépôt GitHub de l'utilisateur `sarathkvn` [5].

Nous voyons cependant dans la Figure 3 que les réseaux neuronaux sont relativement robustes aux perturbations aléatoires, et ce même si les perturbations sont visibles à l'œil nu comme le montre la figure 4.

Cela rejoint les conclusions de Goodfellow et al. [2] qui indiquent que :

La direction de la perturbation, plutôt que le point spécifique de l'espace, est ce qui importe le plus. L'espace n'est pas rempli de poches d'*adversarial examples* qui recouvrent finement les réels comme les nombres rationnels.

Nous pouvons donc voir qu'il faut bien choisir la perturbation pour espérer tromper un réseau neuronal : une at-

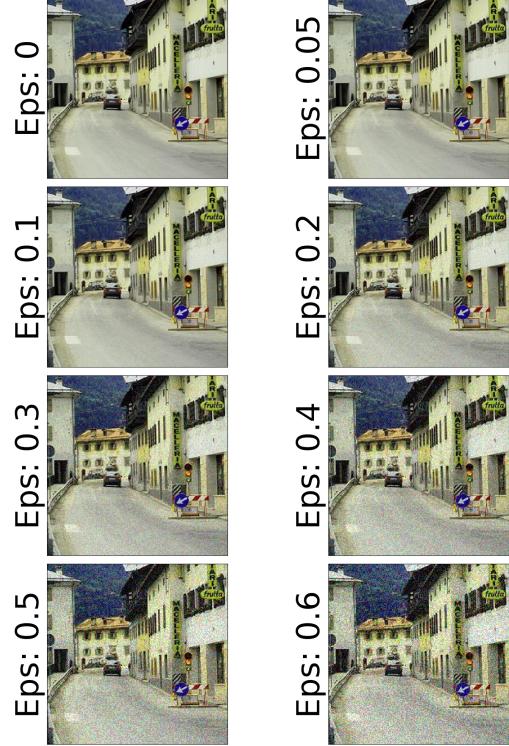


FIGURE 2. Aspect d'une image modifiée avec un masque aléatoire pour différentes valeurs de ϵ

taque ne saurait consister en une simple dégradation de la qualité d'image.

5. Fast Gradient Sign Method

Nous nous intéressons dans cette partie à l'une des premières attaques de réseaux neuronaux, décrite par Goodfellow et al. [2], la *Fast Gradient Sign Attack* (attaque rapide

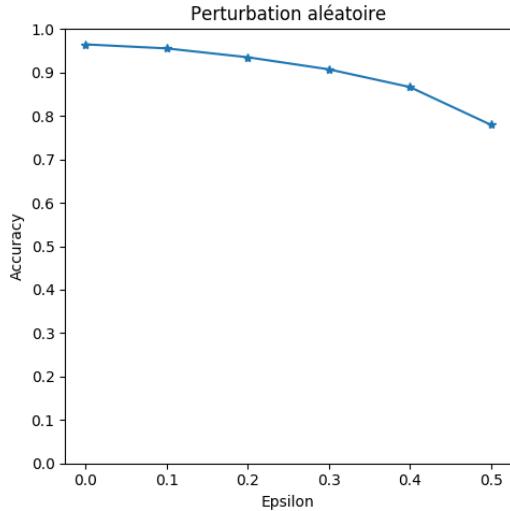


FIGURE 3. Accuracy du réseau neuronal en fonction de ϵ pour une perturbation aléatoire

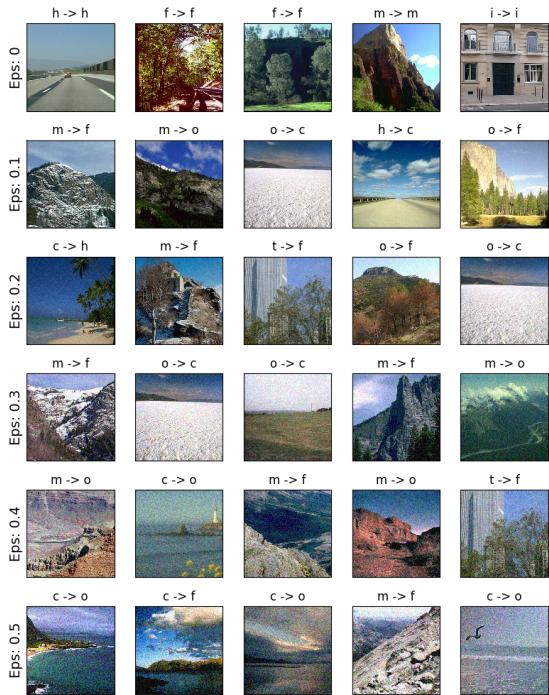


FIGURE 4. Exemples de perturbation aléatoire trompant le réseau neuronal pour différentes valeurs de ϵ

du signe du gradient).

L'idée est simple : les réseaux neuronaux fonctionnant par minimisation de la fonction de coût lors de la descente de gradient, l'attaque cherche à modifier l'image de manière à ce que le coût soit maximisé.

C'est cette attaque qui est mise en œuvre dans la Fi-

gure 1 où $\nabla_x J(\theta, \mathbf{X}, y))$ représente alors le gradient de la fonction de coût J utilisée pour entraîner le réseau. Nous connaissons la fonction de coût initiale, cette attaque est donc bien une attaque « boîte blanche ».

Soient \mathbf{x} et \mathbf{x}^{adv} respectivement l'image d'origine et un *adversarial example* associé. On a alors l'attaque suivante :

$$\mathbf{x}^{adv} = \mathbf{x} + \epsilon \text{sign}(\nabla_x J(\theta, \mathbf{x}, y)) \quad (1)$$

Nous avons mis en œuvre une telle attaque en nous inspirant du tutoriel PyTorch sur la génération d'*adversarial examples* déjà utilisé plus haut. Notre domaine de ϵ s'étend de 0 à 0.03 (un ordre de grandeur de moins que pour les perturbations aléatoires). Le code correspondant est disponible dans `fgsm_pert.py`

Le graphique de l'accuracy en fonction de ϵ en figure 5 nous montre que, même avec d'infimes perturbations, le réseau neuronal perd énormément en précision.

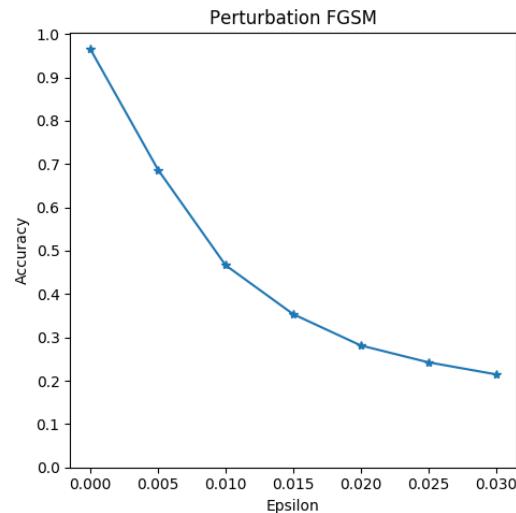


FIGURE 5. Accuracy du réseau neuronal en fonction de ϵ pour la méthode FGSM

Nous avons des exemples d'images modifiées avec succès en figure 6.

Les limites de cette méthode sont doubles :

1. c'est une attaque en boîte blanche, il faut en effet connaître l'architecture du modèle attaqué, et notamment sa fonction de coût ;
2. c'est une attaque qui va réussir à mal classifier les données, mais qui ne va pas permettre de choisir la classe cible.

6. Iterative Fast Gradient Sign Method

Une amélioration de la méthode FGSM est la méthode FGSM itérative [6]. Cette attaque consiste à avancer plu-



FIGURE 6. Exemples de perturbation FGSM trompant le réseau neuronal pour différentes valeurs de ϵ

sieurs fois dans la direction du gradient en s'assurant que chaque pixel reste dans un voisinage de l'image d'origine.

En posant *Clip* la fonction qui s'assure que nous restons dans le voisinage (pixel à pixel) de l'image d'origine par seuillage, on a :

$$\mathbf{x}_0^{adv} = \mathbf{x} \quad \mathbf{x}_{n+1}^{adv} = Clip_{x,\epsilon} \left\{ \mathbf{x}_n^{adv} + \alpha \nabla_x J(\boldsymbol{\theta}, \mathbf{x}_n^{adv}, y) \right\} \quad (2)$$

Le résultat de l'implémentation de cette méthode (adapté des implémentations précédentes) est disponible sur la figure 7. Nous pouvons voir que la méthode itérative est plus efficace que la méthode FGSM simple, car elle réussit à tromper le réseau de manière plus significative pour des perturbations de même poids.

L'attaque est donc plus efficace que le FGSM, au détriment d'un temps de calcul (lié aux N itérations) plus élevé. Elle conserve cependant les deux défauts de FGSM que nous avons identifié.

Le code de cette attaque est disponible dans le fichier `iterative_pert.py`

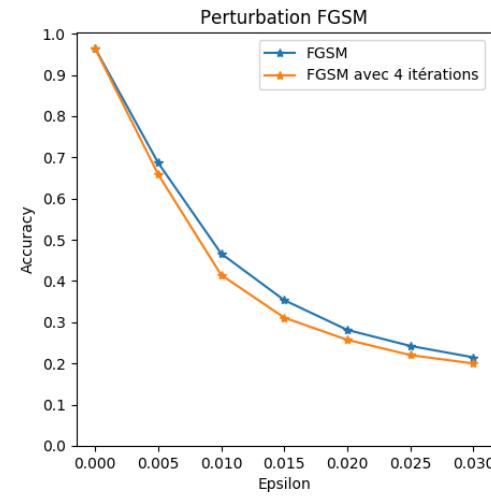


FIGURE 7. Accuracy du réseau neuronal en fonction de ϵ pour la méthode FGSM itérative. Nous avons choisi $N = 4$ itérations et $\alpha = 1$.

7. Attaque ciblée

Un inconvénient majeur des attaques que nous avons étudié (FGSM et FGSM itérative) est l'impossibilité de choisir la classe cible.

Nous pouvons tenter de corriger cela en modifiant l'attaque ainsi pour FGSM :

$$\mathbf{x}^{adv} = \mathbf{x} - \epsilon sign(\nabla_x J(\boldsymbol{\theta}, \mathbf{x}, y_{cible})) \quad (3)$$

Et pour FGSM itérative :

$$\mathbf{x}_0^{adv} = \mathbf{x} \quad \mathbf{x}_{n+1}^{adv} = Clip_{x,c} \left\{ \mathbf{x}_n^{adv} - \alpha \nabla_x J(\boldsymbol{\theta}, \mathbf{x}_n^{adv}, y_{pred}) \right\} \quad (4)$$

Ainsi, intuitivement, au lieu de chercher à aller contre le sens de la descente de gradient en maximisant le coût de la prédiction classe cible, on cherche à minimiser le coût de la prédiction de la classe cible. Pour le choix de la classe cible, nous pouvons choisir la classe que nous désirons. Deux méthodes courantes de sélection de la classe cible sont l'aléatoire ou le choix de la classe la moins susceptible d'être choisie par le réseau pour l'image attaquée [6].

7.1. Attaque avec une cible prédéfinie

Les figures 8 et 9 nous montrent les résultats d'une implémentation de l'attaque FGSM sans itération avec comme cible la classe `mountain`, choisie arbitrairement.

Nous pouvons tout d'abord voir que l'accuracy du réseau neuronal après cette attaque est réduite. Cela est dû à deux choses :

1. Les images de la classe mountain sont correctement classées, ce qui diminue forcément l'*accuracy* même s'il n'y avait pas de diminution sur les autres classes.
2. L'attaque se fait en « allant dans la direction » de la

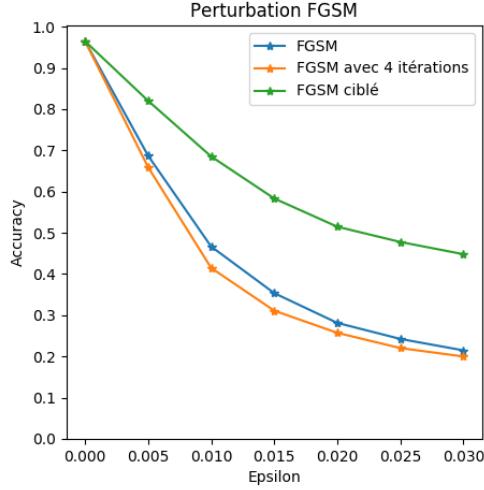


FIGURE 8. Accuracy du réseau neuronal en fonction de ϵ pour les différentes méthodes FGSM.

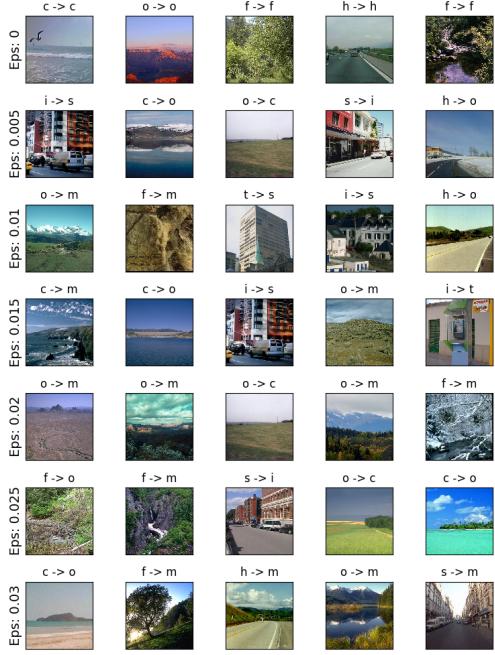


FIGURE 9. Exemples de perturbation FGSM ciblant sur la classe mountain trompant le réseau neuronal pour différentes valeurs de ϵ . Nous pouvons remarquer que toutes les classes des *adversarial examples* ne sont pas mountain.

classe cible. La classe cible étant arbitraire, cela n'est à priori pas aussi efficace qu'aller « dans la direction opposée » à la classe d'origine.

L'attaque classe t-elle l'image dans la classe cible ? Nous pouvons voir sur la figure 10, qui représente le taux d'images classées comme mountain en fonction de ϵ , que bien qu'il y a de plus en plus de classification dans la classe mountain avec l'augmentation de ϵ , toutes les mauvaises classifications ne se font pas dans cette classe.

Nous pouvons voir des exemples de mauvaise classification avec la classe résultante sur la figure 9.

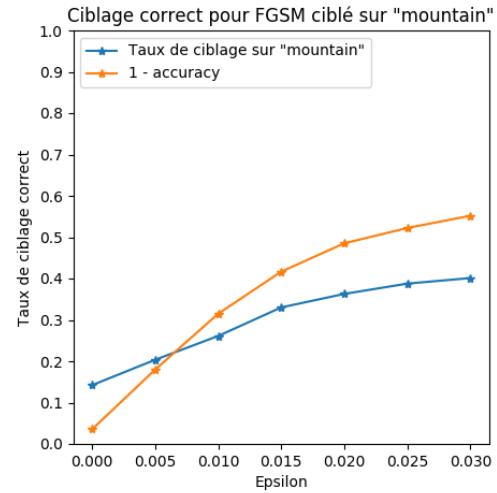


FIGURE 10. Taux de ciblage correct pour une attaque FGSM ciblant sur la classe mountain

7.2. Attaque sur la least-likely class

Cette attaque vise à cibler la classe ayant la plus petite probabilité de représenter notre image d'origine.

Une implémentation de cette méthode donne les résultats des figures 11 et 12.

Nous voyons que la FGSM ciblée *least-likely* et la FGSM ciblée sur une classe arbitraire ont des résultats similaires. Cependant, pour les valeurs de ϵ testées, nous voyons qu'il y a en fait très peu d'images modifiées qui sont classifiées dans la *least-likely class*.

7.3. Code

Le code est disponible dans `fgsm_target_pert.py`. Nous avons aussi écrit une méthode itérative ciblée dans `iterative_target_pert.py`.

8. One Pixel Attack

Les perturbations que nous avons étudiée est facilement détectable : la méthode de perturbation étant l'ajout

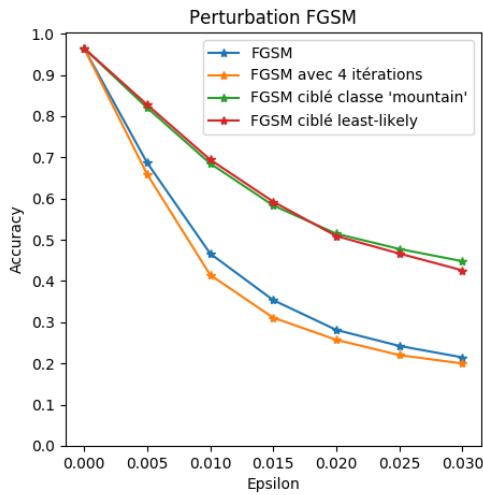


FIGURE 11. Accuracy du réseau neuronal en fonction de ϵ pour les différentes méthodes FGSM.

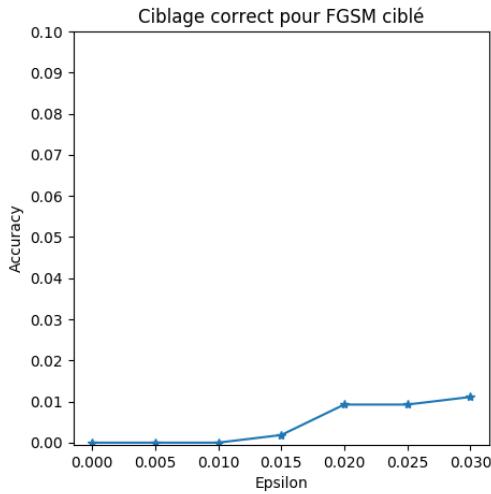


FIGURE 12. Taux de ciblage correct pour une attaque FGSM ciblant sur la *least-likely class*. Notez la différence d'échelle par rapport à la figure 10.

à l'image d'un bruit (bien que spécifique), des méthodes de réduction de bruit peuvent permettre de se prémunir d'une telle attaque [7].

Il existe cependant d'autres manières de générer des *adversarial examples*. L'une d'entre elle, baptisée *One Pixel Attack*, vise à modifier seulement quelques pixels bien choisis [8].

Nous avons essayé de mettre en œuvre cette méthode dans le fichier `onepixel.pert.py` en nous inspirant du code de l'utilisateur GitHub `sarathknn` [5]. Cependant,

l'attaque initiale était effectuée sur le jeu de données CIFAR-10 [9], qui est constituée d'images 32×32 . Notre jeu de données étant une base d'images 256×256 (il faut donc modifier plus de pixels pour réussir l'attaque) et la méthode étant basée sur un algorithme d'évolution différentielle, très coûteuse en ressources, nous n'avons pas pu tester notre implémentation avec succès.

9. Pour aller plus loin

Pour aller plus loin, les pistes d'amélioration suivantes sont proposées :

- Nous avons été grandement limités par la puissance de calcul et le temps disponible (scripts lancés sur un MacBookPro sur un CPU Intel Core i5 @ 2.3GHz). Il serait souhaitable de mener les mêmes études sur des domaines de variables plus étendus (ϵ , α ou nombre d'itérations par exemple).
- Nous avons étudié des méthodes d'attaque (génération d'*adversarial examples*), il serait souhaitable d'étudier aussi des méthodes de défense (ne pas se faire tromper par des *adversarial examples*).
- Nous avons étudié des méthodes d'attaque en « boîte blanche ». Nous n'avons pas toujours accès à l'architecture des réseaux neuronaux, il serait donc souhaitable d'étudier des méthodes d'attaque en « boîte noire ».
- Nous souhaiterions mettre en place d'autres méthodes, dont des GANs.
- Sur un plan plus pragmatique, il serait souhaitable de procéder à un nettoyage de notre code.

10. Conclusion

Nous avons étudié plusieurs méthodes de génération d'*adversarial examples*. Plusieurs points ont été évoqués :

- Les *adversarial examples* ne sont pas un simple ajout de bruit aléatoire.
- Une méthode rapide pour générer un *adversarial example* et la manipulation de la manière dont fonctionne un réseau neuronal : la minimisation d'une fonction de coût. Si on connaît l'architecture du réseau, on peut manipuler notre image afin d'augmenter le coût de la classe d'origine.
- Il est possible de choisir quelle mauvaise classe va être choisie par le réseau neuronal pour une image donnée. Ces méthodes sont cependant moins performantes qu'un simple FGSM.

Références

- [1] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 2013.

- [2] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv :1412.6572*, 2014.
- [3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.
- [4] Aude Oliva and Antonio Torralba. Modeling the shape of the scene : A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3) :145–175, May 2001.
- [5] Sarath Chandra. Implementation of papers on adversarial examples. <https://github.com/sarathknn/adversarial-examples-pytorch>, 2018.
- [6] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.
- [7] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wen-chang Shi, and XiaoFeng Wang. Detecting adversarial image examples in deep neural networks with adaptive noise reduction. *IEEE Transactions on Dependable and Secure Computing*, page 1–1, 2018.
- [8] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, page 1–1, 2019.
- [9] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.