Rapport Projet Sciences des données : Robust neural networks

Maxence Philbert, Tiphaine Le Clercq De Lannoy, Vincent Gouteux

Decembre 2019

Contents

1 Présentation du Problème et des données

Dans ce projet, nous nous intéressons à la robustesse des réseaux de neurones. Nous allons dans un premier temps nous allons implémenter des attaques *adversarielles* pour tromper un réseau de neurones de classification d'images. Nous allons voir deux types d'attaques et comparer leurs effets sur les performances du modèle. Dans une seconde partie nous allons essayer de trouver des moyens pour rendre les réseaux robustes a ce genre d'attaques notamment grâce a l'*adversarial training*. Finalement nous allons voir comment est il possible de tomper un réseau quand on ne sait rien de ce dernier, c'est a dire des attaques types *Black-Box*. .

Les données :

Le jeu de données que nous allons utiliser est le CIFAR10 composé de 60 000 images de taille $32 * 32$ ce jeu de donnée est extrêmement utilisé dans les problèmes de classification d'images car il n'est pas très volumineux et est donc facile a manipuler de par sa simplicité. Il se compose donc de 60 000 images reparties en 10 classes : Avion, Voiture, Oiseau, Chat, Chien, Cerf, Grenouille, Cheval, Bateau, Camion

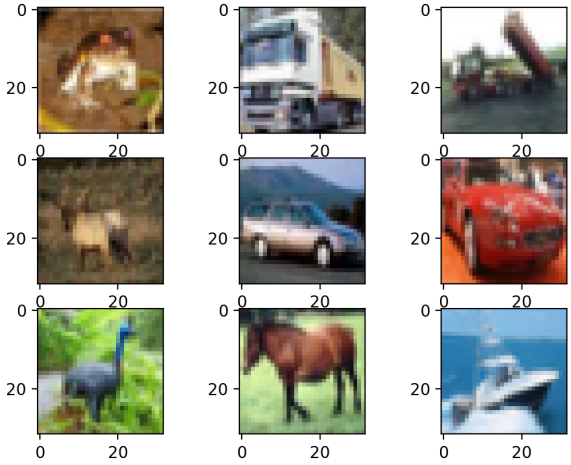


Figure 1: Exemples d'images CIFAR10

2 Attaques L^∞ et L^2

Dans cette première partie nous allons voir ce qu'est une attaque adversarielle dans le cas d'un réseau de neurones de classification d'images. Nous allons en implémenter deux méthodes : *Fast Gradient Signed Method (FGSM)* et *Projected Gradient Descent (PGD)*.

Une attaque adversarielle vise a générer des exemples dont le but sont de tromper un réseau de neurones. Cela permet d'exploiter les "failles" du réseau et de tester ses limites. Dans le cas précis des réseaux de classification d'images une attaque adversarielle consiste a demander au réseau de prédire la classe d'une image a laquelle on rajoute une certaine perturbation. La perturbation n'est bien sûr pas prise au hasard. Le but de ces attaques va être de déterminer quelles sont les perturbations optimales qui trompent le réseau de neurones. La contrainte pour que ces attaques soient pertinentes est que la norme de la perturbation doit être bornée, c'est-à-dire que la perturbation doit être assez faible pour ne pas se faire détecter par un humain mais doit faire en sorte que le réseau prédise la mauvaise classe.

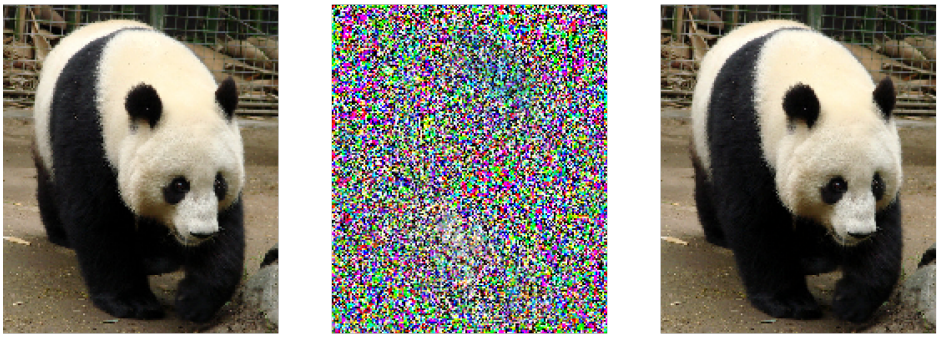
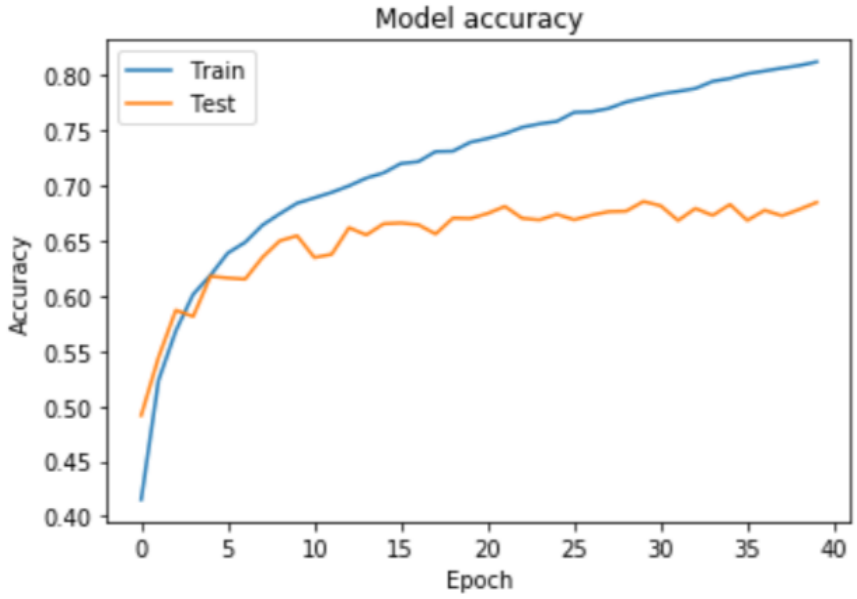


Figure 2: L'image perturbée semble identique pour l'humain mais la machine la classifie mal

2.1 Le réseau

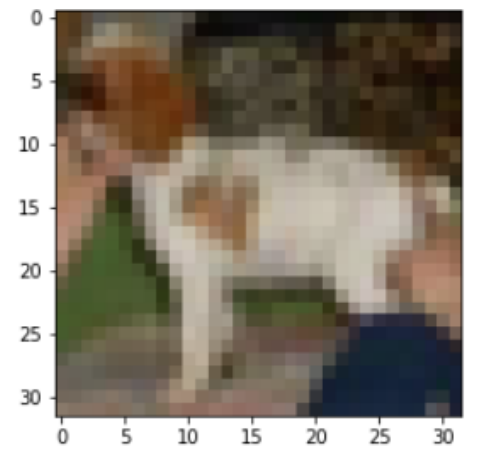
Le réseau que nous allons implémenter pour effectuer nos attaques sera un réseau assez simple de classification d'image dont l'architecture est la suivante : $(Conv + ReLU + MaxPool) * 3 + AvgPool + FC + ReLU + FC + SoftMax$. Nous l'avons implémenté sur keras en passant par l'API tensorflow. Après plusieurs tests nous avons determiné que 40 epochs étaient suffisantes pour apprendre sans overfitter. Nous obtenons les performances suivantes :




```
Train accuracy : 0.81235 Test accuracy :  0.6848
```

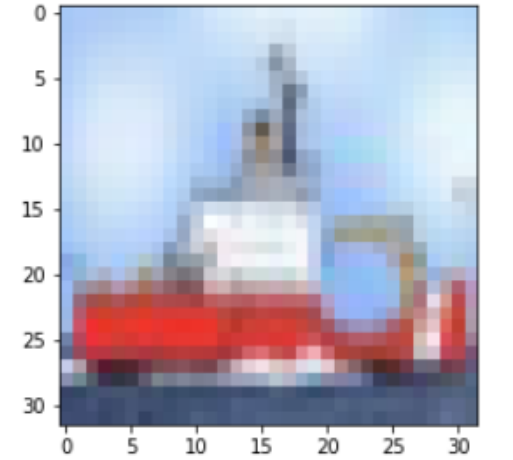
Figure 3: Précision de modèle en fonction du nombre d'epochs

La précision sur l'echantillon de test monte au alentours de 70% ce qui est satisfaisant pour des images de cette qualité. Nous pouvons verifier avec deux exemples :



```
Prediction (Orginal image) :  dog
```

(a) Image de chien bien classifiée



```
Prediction (Orginal image) :  ship
```

(b) Image de bateau bien classifiée

Figure 4: Exemples de classifications d'images non perturbées

2.2 Attaques FGSM et FGM

Soit $f : \mathbb{R}^n \to \mathcal{Y}$ le classifieur qui à chaque image x_i associe son label/classe y. Dans notre cas $x_i \in \mathbb{R}^{32*32*3}$ et $y \in \{1, \dots, 10\}$.
Le but va être de générer une perturbation δ de taille $32 * 32 * 3$ telle que $f(x) = y$ mais $f(x + \delta) \neq y$ sous la contrainte $||\delta|| \leq \epsilon$ Le but est donc de trouver la perturbation qui maximise la fonction de perte en $x + \delta$ c'est-à-dire trouver δ tel que :

$$\delta = \underset{||\delta|| \leq \epsilon}{argmax} \mathcal{L}(f, x + \delta, y)$$

Nous allons voir dans cette partie deux types d'attaques de *Fast Gradient Method* mais nous allons utiliser deux normes différentes pour borner la norme du gradient

2.2.1 L^∞ : Fast Gradient Signed Method

On rappelle que la norme infinie se calcule de la façon suivante : $||\delta||_\infty = \underset{i}{max}|\delta_i|$ On veut trouver la perturbation optimale telle que

$$\delta = \underset{||\delta||_\infty \leq \epsilon}{argmax} \mathcal{L}(f, x + \delta, y)$$

pour ϵ petit ce problème revient à résoudre

$$\delta = \underset{||\delta||_\infty \leq \epsilon}{argmax} \delta^t \nabla_x \mathcal{L}(f, x, y)$$

On sait par le papier de Ian J. Goodfellow, Jonathon Shlens Christian Szegedy que ce problème de maximisation possède une solution qui est

$$\delta^* = \epsilon sign(\nabla_x \mathcal{L}(f, x, y))$$

2.2.2 L^2 Fast Gradient Descent

. On rappelle que la norme L^2 se calcule de la façon suivante : $||\delta||_2 = \sqrt{\sum_i \delta_i^2}$. Ici le problème est similaire, seulement la norme pour borner la perturbation est différente

$$\delta = \underset{||\delta||_2 \leq \epsilon}{argmax} \mathcal{L}(f, x + \delta, y)$$

Dans ce cas la perturbation optimale est de la forme

$$\delta^* = \epsilon \frac{\nabla_x \mathcal{L}(f, x, y)}{||\nabla_x \mathcal{L}(f, x, y)||_2}$$

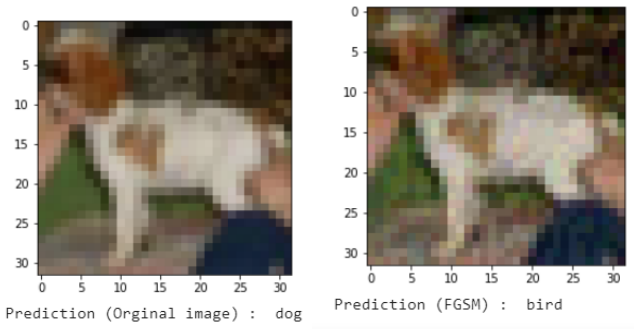
Pour plus de détails sur les calculs voir le papier suivant : Boosting Adversarial Attacks with Momentum.

2.2.3 Résultats

En fonction du ϵ choisi pour borner la norme de la perturbation, on peut arriver à des résultats extrêmement satisfaisants ; en effet le réseau se trompe quasiment à chaque fois sur les images perturbées là où il ne se trompait que 30% du temps. Nous pouvons donc comparer les résultats des performances dans le tableau ci dessous :

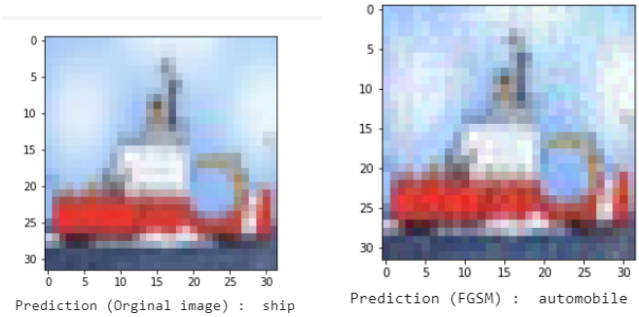
	Images originales	Attaque FGSM $\epsilon = 0.031$	Attaque FGM $\epsilon = 0.4$
Training Accuracy	0.812	0.0988	A 0.0982
Testing Accuracy	0.684	0.0014	0.1303

Table 1: Précisions d'entraînement et de validations obtenues



(a) Chien (b) Oiseau

Figure 5: Mauvaise classification d'une image de chien



(a) Bateau (b) Voiture

Figure 6: Mauvaise classification d'une image de bateau

2.2.4 Test sur ϵ

Pour qu'une attaque adversarielle soit pertinente il faut que la perturbation appliquée soit assez importante pour tromper le réseau mais pas visible pour un humain. On contrôle la perturbation par sa norme que l'on borne par un certain ϵ , l'idée est donc de trouver un ϵ de sorte à tromper efficacement le réseau sans que l'image ne semble différente. On peut donc, pour chaque image, générer les perturbations optimales (FGSM) pour différents ϵ, et tester les prédictions du modèle.

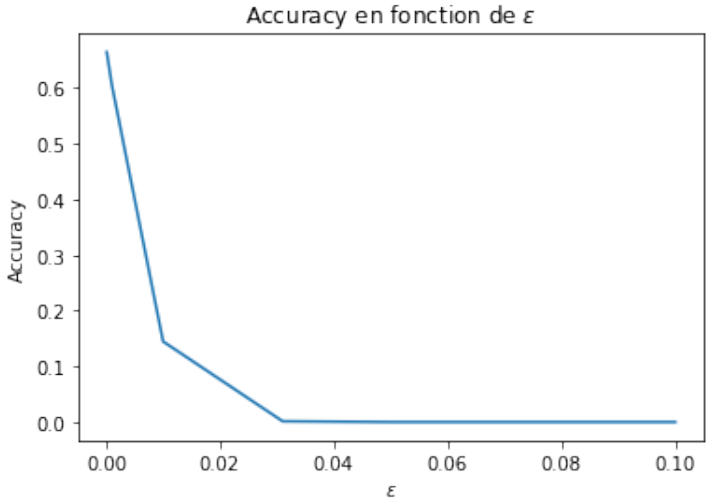
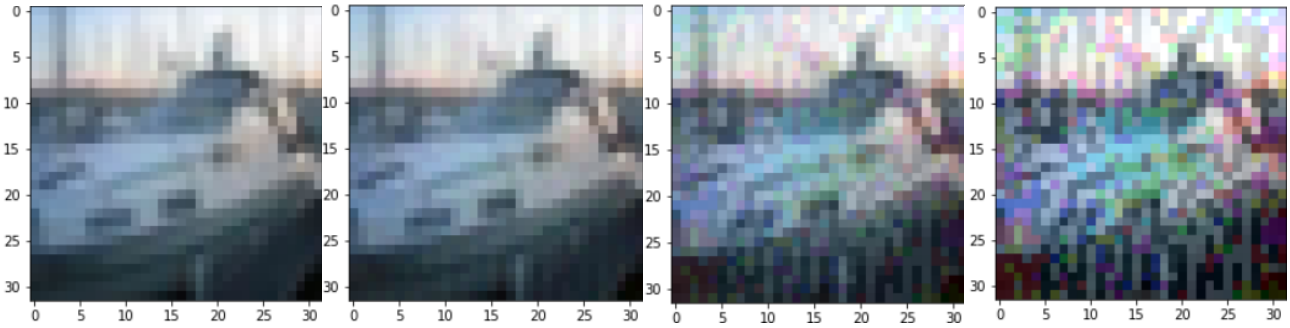


Figure 7: Accuracy en fonction de ϵ



(a) Orignial (b) $\epsilon = 0.01$ (c) $\epsilon = 0.05$ (d) $\epsilon = 0.1$

Figure 8: Images perturbées pour différents ϵ

Certains papiers de recherche sur le sujet ont établi qu'il existait des bornes optimales pour la norme de la perturbation : $\epsilon_{\ell\infty} = 0.031$ et $\epsilon_{\ell 2} = 0.4$ Nous avons donc utilisé ces ϵ pour effectuer tous nos tests.

2.3 Attaques PGD

L'attaque PGD (*Projected gradient descent*) est une version itérative des attaques FG(S)M vues précédemment. On génère itérativement des perturbations de plus en plus importantes (pendant un certain nombre d'itérations), de manière à maximiser notre fonction de perte, tout en veillant à rester dans une boule de rayon ϵ. Autrement dit, en notant x_t notre image perturbée à l'itération t, l'algorithme est le suivant :

$$\begin{cases} x_0 &= x \\ x_{t+1} &= \pi_{B(0,\epsilon)}(x_t + \underset{||\delta|| \leq \eta}{argmax} \mathcal{L}(f, x_t + \delta, y)) \end{cases}$$

où $\pi_{B(0,\epsilon)}$ désigne le projeté sur une boule de rayon ϵ

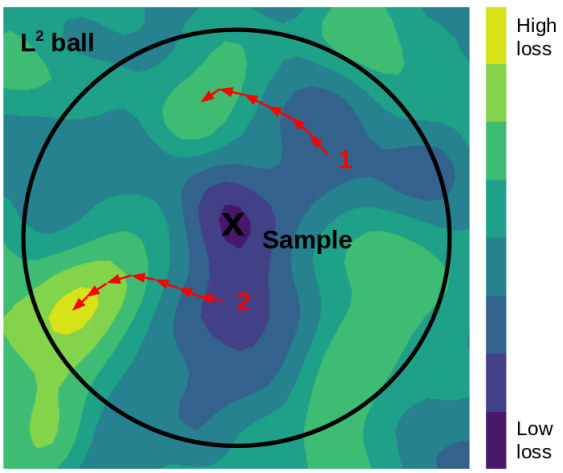


Figure 9: Attaque PGD

Nous pouvons, ici également, utiliser deux normes différentes :

2.3.1 Norme L^∞

Avec la norme infinie $||\delta||_\infty = \underset{i}{max}|\delta_i|$, l'attaque PGD revient à générer x_t de la manière suivante :

$$\begin{cases} x_0 &= x \\ x_{t+1} &= \pi_{B(0,\epsilon)}(x_t + \eta sign(\nabla_x \mathcal{L}(f, x_t, y))) \end{cases}$$

2.3.2 Norme L^2

Avec la norme L^2 $||\delta||_2 = \sqrt{\sum_i \delta_i^2}$, l'attaque PGD revient à générer x_t de la manière suivante :

$$\begin{cases} x_0 &= x \\ x_{t+1} &= \pi_{B(0,\epsilon)}(x_t + \epsilon \frac{\nabla_x \mathcal{L}(f,x_t,y)}{||\nabla_x \mathcal{L}(f,x_t,y)||_2}) \end{cases}$$

2.3.3 Résultats

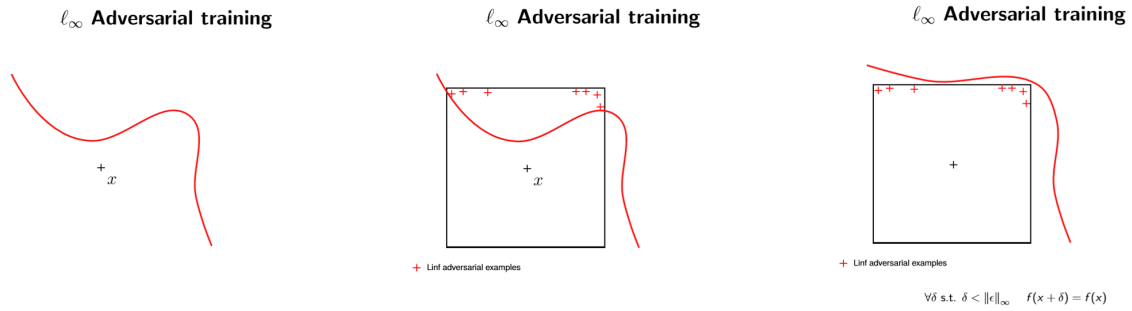
On choisit un nombre d'itérations égal à 20.

	Images originale	PGD L^∞ ($\eta = 0.05$ et $\epsilon = 0.031$)	PGD L^2 ($\eta = 0.05$ et $\epsilon = 0.4$)
Training Accuracy	0.812	0.0052	0.0044
Testing Accuracy	0.684	0.1465	0.0906

Table 2: Précision d'entraînement et de validations obtenues

3 Adversarial training

Dans cette partie nous allons voir une défense possible contre ce genre d'attaques : *l'Adversarial training*. Cette méthode consiste à entraîner le réseau avec des images auxquelles on a ajouté une perturbation. Comme étudié dans la partie précédente, nous savons que les perturbations sont calculées d'une façon précise. Pour apprendre au réseau à se défendre nous allons donc à chaque epoch calculer le gradient et la perturbation. Nous allons ensuite appliquer la perturbation à notre ensemble d'entraînement et ré-entraîner le réseau avec les images perturbées. Nous allons répéter cette méthode sur une dizaine d'epochs jusqu'à obtenir une accuracy satisfaisante.
Les trois images ci dessous schematisent les principales étapes de *l'Adversarial training* Source : B.Negrevergne, L.Meunier



(a) Frontière originale (b) Création d'attaques (c) Entraînement sur images perturbées

Figure 10: Étapes de *l'Adversarial Training*

3.1 Résultats

Nous avons donc réalisé de *l'Adversarial training* pour les deux types d'attaques vues précédemment. L'architecture du réseau reste, elle, inchangée. Nous avons d'abord implémenté un entraînement sur des images perturbées par FGSM puis un autre réseau identique entraîné celui ci avec des images perturbées par PGD.

		Image originale	FGSM	PGD
Réseau Original	*Training Accuracy*	0.812	0.0988	0.0982
	Testing Accuracy	0.684	0.0014	0.1303
Réseau FGSM	*Training Accuracy*	-	-	-
	Testing Accuracy	0.4321	0.3618	0.3683
Réseau PGD	*Training Accuracy*	-	-	-
	Testing Accuracy	0.5863	0.3823	0.4048

Table 3: Précisions obtenues pour les 3 modèles

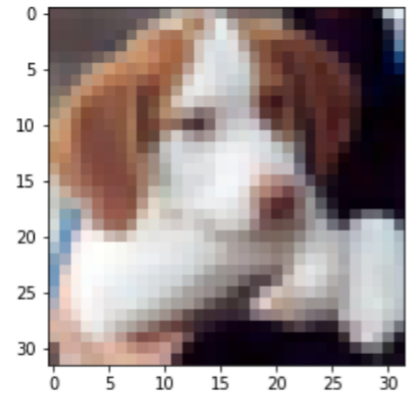
On remarque des résultats assez mitigés. On observe que là où on arrivait très bien à tromper le réseau (le PGD était devenu aussi mauvais que l'aléatoire, et le FGSM se trompait quasiment à chaque fois soit **pire qu'un classifieur aléatoire**). Ici pour les deux réseaux entraînés contre les attaques on remarque que les performances sont bien meilleures (environ 40% de précision) donc même si le classifieur reste mauvais, il apprend à se défendre contre les attaques. On remarque que le classifieur FGSM se défend avec la même efficacité contre les deux types d'attaques. A l'inverse, le réseau PGD se défend mieux contre les attaques PGD que FGSM ce qui parait cohérent même si cette différence n'est pas très significative.
Cependant même si les réseaux arrivent plus ou moins à se défendre contre les attaques ils perdent en précision sur les images originales. Ce résultat est regrettable car en essayant de protéger le réseau contre des attaques adversarielles on se retrouve à le faire se tromper sur des images ou il prédisait la bonne classe au paravent. On peut cependant améliorer ces résultats en entraînant le réseau plus longtemps et en rajoutant les images originales dans le jeu d'entraînement a chaque epoch. Bien que l'on perde un petit peu en précision sur les images non perturbées, on arrive un petit peu a classifier des images perturbées la ou le réseau se trompait quasiment systématiquement.

4 Attaques Black-box

Dans cette dernière partie l'approche est assez différente : on suppose que l'on ne sait rien sur le réseau, seulement sa prédiction pour une entrée donnée. Nous n'avons donc pas accès au gradient de la fonction de perte et nous ne pouvons pas effectuer d'attaques type FGSM ou PGD vues précédemment. Nous avons vu dans les partie précédentes que les attaques n'étaient pas faites au hasard, bien au contraire, les perturbations avec une direction très précise calculée dans l'unique but d'augmenter la fonction de perte et donc de tromper le réseau. Ici la grande difficulté sera donc de trouver la direction de la perturbation optimale pour tromper le réseau tout en faisant en sorte que l'image modifiée semble identique a l'image de base.

4.1 Générateur de perturbations aléatoires

La seule donnée dont on dispose est la prédiction du réseau. Cette prédiction est sous forme d'un vecteur de taille 10 ou chaque valeur symbolise la probabilité que l'image appartienne à une certaine classe. Ce sont ces probabilités que nous allons utiliser pour générer les perturbations. Comme on peut voir ci dessous le réseau prédit que l'image est un chien a 88.3%



```
[[0.    0.    0.018 0.077 0.    0.883 0.009 0.011 0.    0.   ]]
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

Figure 11: Prédiction de l'image par le réseau

Nous avons donc créé un générateur de perturbations aléatoires qui va essayer de tromper le réseau. Le principe est de générer une perturbation aléatoire très faible et de regarder ce que prédit le réseau. Si la probabilité d'appartenir à la bonne classe diminue alors cela veut dire que notre perturbation a eu l'effet désiré sur le réseau nous conservons donc cette image perturbée si en revanche la probabilité augmente alors nous ne conservons pas cette perturbation et nous retournons à l'image à l'*état* précèdent.

Nous bouclons ainsi de suite en rajoutant ou non des perturbations aléatoires. Si tout se passe bien, la condition d'arrêt est que le réseau s'est trompé et donc que nous avons réussi à générer une bonne perturbation. En revanche il est possible que les perturbations ne soient pas bonne et donc que le réseau continue de prédire la bonne classe. Dans ce cas nous avons ajouté une autre condition d'arrêt qui est que nous nous éloignons trop de l'image originale et que donc la perturbation serait visible. Cette condition d'arrêt est simplement une borne sur le norme de la perturbation. Le choix de la borne est assez subjectif. Nous avons effectués plusieurs tests afin de décider de la borne, c'est à dire à partir de quelle norme de perturbation cette dernière devenait visible.
En étant étant assez restrictif sur la norme de la perturbation pour que les attaques soient cohérentes nous obtenons des résultats assez satisfaisants. Pour mesurer la pertinence de cette méthode nous essayons de générer des perturbations pour chaque image. Si nous ne parvenons pas à tromper le réseau avec des perturbations de normes inférieures à la borne fixées nous le comptons comme un échec. La précision de cette méthode sera donc :

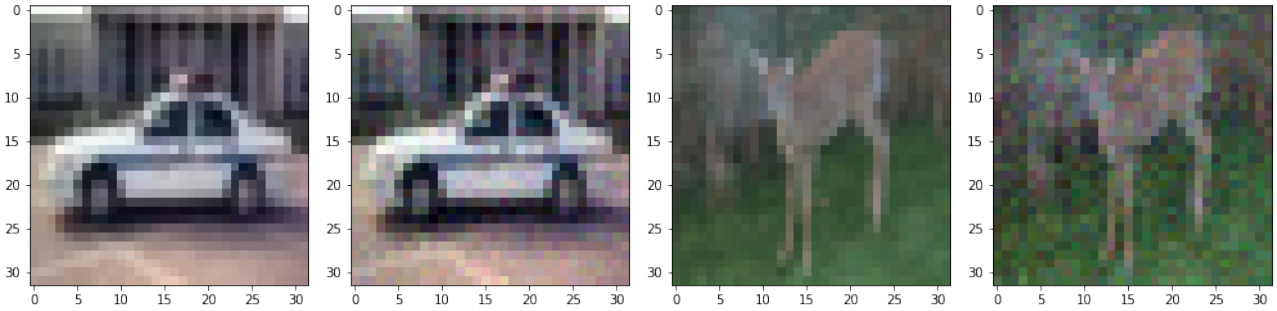
$$1 - \frac{\#_{échecs}}{\#_{total}}$$

	Images originales	Attaque Random
Testing Accuracy	0.684	0.125

Table 4: Précision du modèle sur les images avec perturbations *Aléatoires*

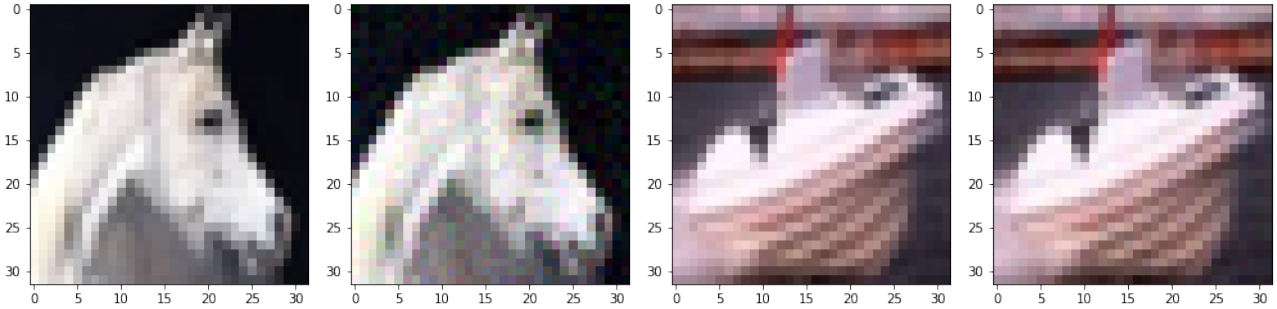
```
[200] print(adversarial.evaluate(np.reshape(x_pert,[1000,32,32,3]), y_test[:1000]))

1000/1000 [==============================] - 0s 85us/sample - loss: 1.3478 - accuracy: 0.1250
[1.3477889013290405, 0.125]
```

Figure 12: Prédiction des images par le réseau



(a) Voiture (b) Camion (c) Cerf (d) Cheval

Figure 13: Images perturbées *Aléatoirement* et leur prédictions

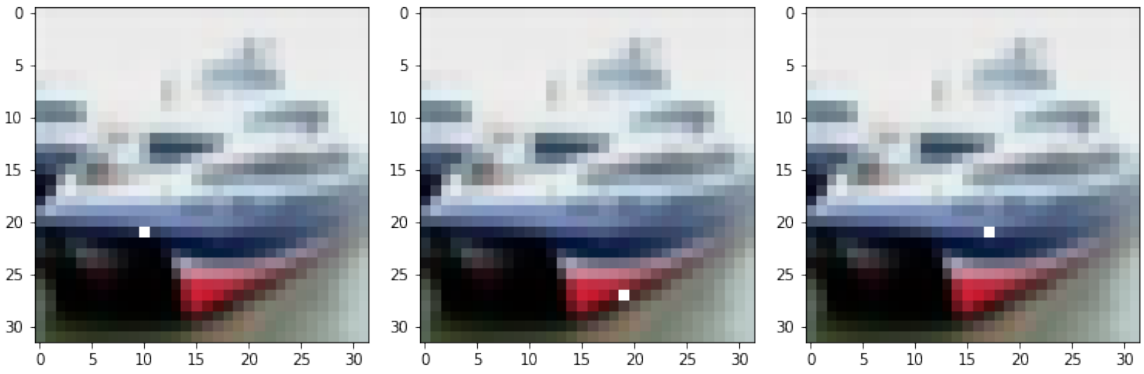


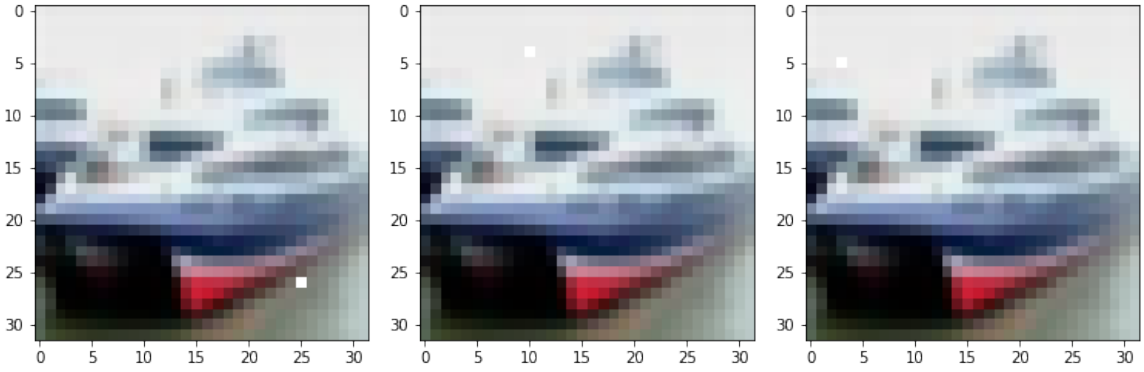
(a) Cheval (b) Oiseau (c) Bateau (d) Chat

Figure 14: Images perturbées *Aléatoirement* et leur prédictions

4.2 Modification d'un pixel

Nous avons également remarqué qu'il était possible de tromper le réseau en ne modifiant qu'un seul et unique pixel. Il existe des méthodes bien précises pour trouver la perturbation idéale c'est à dire quel pixel et quelle couleur lui affecter. Nous n'avons pas implémenté ces méthodes mais nous avons effectués quelques tests pour mettre en évidence les failles de ces réseaux. Le test que nous avons effectué est sur une seule et même image nous avons demandé au réseau de prédire la même image 1024 fois mais a chaque fois avec un pixel blanc. Les résultats sont très encourageants car quand nous demandons au réseau de prédire la classe de l'image sans modification il trouve 100% de temps (peu importe combien de fois on lui demande) En revanche pour les pixels, il s'est trompé 298 fois. Soit environ 30% de mauvaise classification. Pour la modification d'un seul pixel ces résultats sont très surprenants.





(a) Camion (b) Camion (c) Camion

Figure 16: Exemples de pixels qui provoquent une mauvaise classification

On peut voir sur les images ci-dessus que bien entendu la position du pixel blanc est importante car sur les 1024 positions possibles seules 298 ont réussi a tromper le réseau. ce qui est surprenant c'est qu'en tant qu'humain on pourrait penser que les pixels les plus importants sont ceux sur le bateaux ou l'objet en question mais il semblerait que lorsque l'on modifie un pixel dans le *décors* on peut également tromper la machine ce qui prouve encore les failles des réseaux de neurones et leur *réflexion* très différente de la notre.

Conclusion

Dans ce projet nous avons pu constater les failles de certains réseaux de neurones. Nous avons vu avec les attaques FGSM et PGD qu'il est très simple de générer des attaques efficaces. Quand nous avons accès au réseau et donc au gradient de la fonction de perte nous pouvons par une simple formule créer des attaques. Nous avons vu dans le cas des attaques *Black box* nous n'avions pas de formule explicite mais même en générant des perturbations *aléatoires* il était assez simple de tromper un réseau. Une idée de piste qui pourrait être explorée est de générer les perturbations pour toutes les images et entraîner un modèle qui a chaque image associe sa perturbation pour essayer de retomber sur le gradient de la fonctions de perte.Nous avons également vu un moyen de défense qui peut être assez efficace si on dispose de jeux d'entraînements importants cependant nous avons vu qu'il existe encore des failles car même en entraînant le réseau avec des images perturbées il reste très vulnérable a d'autres attaques.