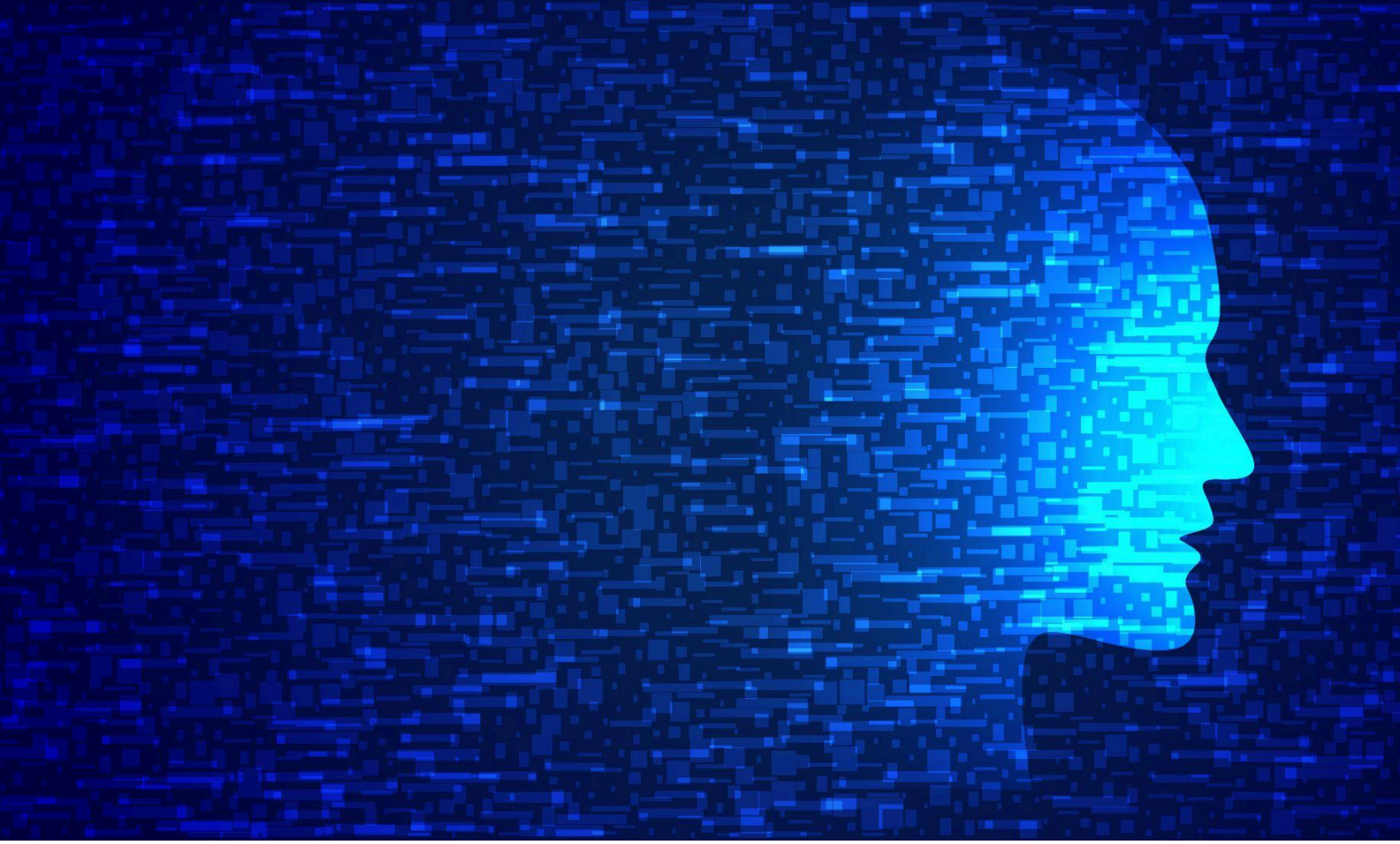


Master de Cybersécurité
Rapport de projet - Semestre 3

Attaques adversariales sur les systèmes de détection de Deepfakes

Cindy Hartmann - Antoine Montier

Novembre - Décembre 2025

Encadré par Christophe Charrier et Emmanuel Giguet

GREYC

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

UNICAEN
UNIVERSITÉ
CAEN
NORMANDIE

Sommaire

Avant-propos

Ce rapport s'inscrit dans le cadre du projet de troisième semestre de Master de Cybersécurité, réalisé courant novembre et décembre 2025.

Il a été encadré par Christophe Charrier, responsable de l'équipe *SAFE* (Sécurité, Architecture, Forensique, biomEtrie) au *GREYC* (Groupe de recherche en informatique, image et instrumentation de Caen), et Emmanuel Giguet, chargé de recherches au *CNRS* affecté au *GREYC*.

Avant toute chose, nous tenons à les remercier pour leur accompagnement et leurs précieux conseils tout au long de ce travail.

Introduction

Avec l'essor de l'Intelligence Artificielle et sa démocratisation auprès du grand public ces dernières années, il est à la portée de tous de générer et de manipuler des contenus multimédias. Ces technologies atteignent aujourd'hui un niveau suffisant pour produire aisément des images, sons ou des vidéos d'un réalisme capable de tromper les sens humains.

C'est dans ce cadre que les deepfakes s'inscrivent. Pour reprendre la définition donnée par la loi européenne sur l'intelligence artificielle (article 3 point 60 du Règlement (UE) 2024/1689, dit "AI Act"), un deepfake désigne une image, un contenu audio ou vidéo généré ou manipulé par IA, présentant une forte ressemblance avec des personnes, des objets, des lieux, des entités ou encore des événements réels, et qui pourrait apparaître faussement authentique ou vrai pour une personne.

Les deepfakes sont aujourd'hui utilisés dans des contextes légitimes tels que le cadre du divertissement ou de la reconstitution historique. Cependant, ils peuvent également être utilisés à des fins malveillantes, comme la diffusion de contenus trompeurs, la désinformation ou encore l'usurpation d'identité.

En 2024, un deepfake du directeur financier d'une entreprise a été généré à partir d'extraits de conférences disponibles publiquement sur Youtube. Ce deepfake a ensuite été immiscé dans une visioconférence, demandant d'effectuer un transfert d'argent vers un compte. Le préjudice a été d'environ 24 millions d'euros.

Cet exemple, ciblant la multinationale *Arup*, basée à Hong-Kong n'est pas un cas isolé. D'après le rapport sur la fraude d'identité publié par SumSub en 2024[1], on parle d'une augmentation à l'échelle mondiale, de 400% du nombre de deepfakes détectés entre 2023 et 2024. Les deepfakes sont utilisés dans 7% des tentatives de fraude, ciblant autant les particuliers que les entreprises.

L'emploi mal intentionné de cette technologie inquiète. En effet, lors d'un sondage publié par le même organisme en 2024, 81% des questionnés affirmaient être soucieux de l'impact des deepfakes sur l'intégrité électorale.

Face à ces risques émergents, la mise en place de systèmes capables de détecter les deepfakes est devenu un enjeu majeur. Ces systèmes visent à confirmer

[1] Sumsub. (2024). Identity fraud report 2024. Disponible sur Scribd : https://www.scribd.com/document/909970242/Sumsub-Identity-Fraud-Report-2024

l'authenticité des contenus, en identifiant les traces laissées par les processus de génération et manipulation artificielle. Ils jouent un rôle important vis-à-vis de la protection de l'information et de la réputation des individus, ainsi que dans la confiance accordée aux médias numériques.

L'intelligence artificielle s'est imposée comme approche privilégiée pour la conception de systèmes de détection de deepfake. Néanmoins, ceux-ci ne sont pas infaillibles et il existe diverses manières de les contourner.

Il est par exemple possible d'ajouter des perturbations minimes pour tromper le détecteur sans dégrader le contenu de manière visible. C'est le principe des attaques adversariales.

Notre projet s'inscrit dans ce contexte, et se concentre justement sur l'étude des attaques adversariales appliquées aux systèmes de détection de face swapping, une technique de deepfake consistant à apposer le visage d'une personne sur une autre.

Nos objectifs étaient les suivants :

- Comprendre le fonctionnement des attaques adversariales;
- Appliquer ces attaques à un modèle de détection de deepfakes / face swapping existant ;
- Mesurer la dégradation des performances du détecteur.

Pour les atteindre, nous avons dans un premier temps effectué un état de l'art des attaques adversariales. Puis, nous avons choisi un modèle et un jeu de données de référence. Enfin, nous avons appliqué certaines de ces attaques sur notre modèle et constaté la dégradation de ses performances.

Nous détaillerons chacune de ces étapes dans le présent rapport.

I• État de l'art des attaques adversariales

La première phase de notre projet a été consacrée à la rédaction d'un état de l'art concernant les attaques adversariales. Nous avons commencé par nous documenter sur les pratiques et techniques de ce domaine.

Au total, nous avons étudié cinq types d'attaques dans notre état de l'art : la *Fast Gradient Signed Method* (*FGSM*), la *Projected Gradient Descent* (*PGD*), *DeepFool* et les *Universal Adversarial Perturbations* (*UAP*) ainsi qu'une dernière, non présentée ici. Ces attaques fonctionnent sur des modèles dits en *"white-box"* : nous pouvons accéder aux paramètres du modèle ainsi qu'aux gradients, c'est-à-dire à la direction au sens mathématique dans laquelle changer les pixels de l'image pour minimiser ou maximiser l'erreur de prédiction.

L'attaque la plus simple à comprendre, *FGSM* fut notre premier pas dans ce domaine. Nous l'avons testé avec le but de comprendre et d'implémenter une attaque typique. Il est facile de trouver des informations concernant *FGSM*. C'est pour cette raison que nous l'avons privilégié. La seconde attaque, *PGD*, est une version plus poussée de *FGSM*. Cela nous a familiarisé avec la manière d'implémenter et de modifier le comportement de ces techniques. La troisième attaque que nous avons étudiée est *DeepFool*. C'est une attaque plus complexe à implémenter. Elle est connue pour son efficacité dans le domaine des deepfakes. Finalement, nous avons étudié les familles d'attaques *UAP*. Ces techniques se démarquent en calculant une unique perturbation universelle qui pourra ensuite être appliquée aux données souhaitées. L'étude des *UAP* nous a permis de renforcer encore nos connaissances en intégrant des attaques déjà développées afin de calculer cette perturbation.

Pour chaque attaque, nous avons d'abord étudié leur principe et leur algorithme en nous appuyant sur des publications scientifiques et des dépôts *Github*. Ensuite, nous les avons implémentés nous-mêmes, ce qui nous a permis de mieux saisir leurs différents résultats.

Pour cela, nous avons travaillé sur un environnement collaboratif en ligne : *Deepnote*. Nous avons utilisé la librairie *Tensorflow* de python avec les images du *MNIST*, des chiffres dessinés à la main. Nous avons créé un modèle de classification simple mais performant sur ces images : son *accuracy*, c'est-à-dire la proportion d'images correctement prédites, est supérieure à 97%. L'ensemble constitué du

modèle et du jeu d'images a été notre base pour implémenter et tester ces attaques. La figure ci-dessous montre les prédictions du classifieur sur ces images. Le titre de chaque chiffre présente d'abord le chiffre vrai (*'True'*) puis la prédiction du modèle (*'Pred'*) ainsi que sa probabilité.

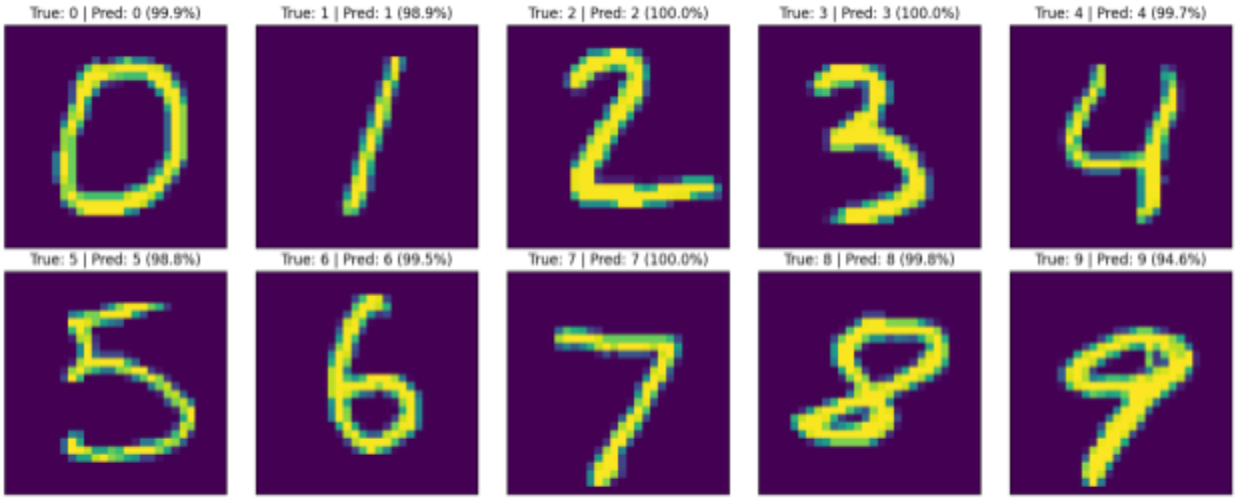


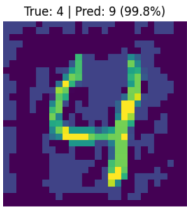
Figure 1. Exemples d'images du jeu de données *MNIST*

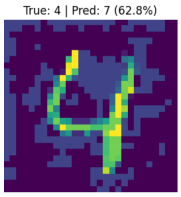
Ce que vous allez lire est une version synthétisée de l'état de l'art, nécessaire pour la compréhension globale de ce rapport. La version complète de l'état de l'art, les implémentations ainsi que les documents consultés sont disponibles en annexe.

a/ *FGSM*

Principe

L'attaque *Fast Gradient Sign Method* (*FGSM*) consiste à générer une image adversariale en ajoutant un bruit calculé à l'aide de la dérivée du coût du réseau. C'est une fonction mathématique mesurant à quel point la prédiction est mauvaise par rapport à la réponse attendue. L'idée est de *"pousser"* les pixels de l'image dans la direction qui augmente le coût si on souhaite tromper au maximum le modèle. L'attaque est dite *"non ciblée"* (figure de gauche, l'image est prédite comme étant un 9 avec une probabilité de 99.8%). On peut également procéder de manière à leurrer le modèle dans un sens particulier (figure de droite, faire prédire une image de *4* comme





étant un 7. L'image est prédite comme étant un 7 avec une probabilité de 63%). Dans ce cas-là, l'attaque est dite *"ciblée"*, on cherche alors à diminuer le coût de la classe cible.

Implémentation

Cette attaque est assez simple à implémenter. En effet, on calcule la dérivée du coût par rapport à l'image en entrée. On crée alors la perturbation qui va dans ce sens. On multiplie ensuite cette perturbation par un facteur d'intensité en fonction du résultat souhaité. Finalement, on ajoute la perturbation à l'image pour obtenir l'exemple adversarial.

Le seul paramètre de cette technique est le facteur d'intensité de la perturbation, noté ϵ. Sa valeur, entre 0 et 1 permet de jouer avec l'efficacité et la visibilité de l'attaque. Réglé proche de 0, l'attaque sera peu visible, mais sans efficacité garantie. D'un autre côté, pour des valeurs avoisinant 1, l'attaque sera très visible mais très efficace. Ce réglage est une manière de comprendre le dilemme discrétion/efficacité des attaques. La figure ci-contre présente le taux de précision (proportion d'images perturbées ayant trompé le modèle) des deux versions de l'attaque pour des valeurs de ϵ comprises entre 0 et 0.3. Ce graphique a été réalisé à partir de 96 images. On remarque que la version non ciblée (bleu) sature pour un ϵ supérieur à 0.17.

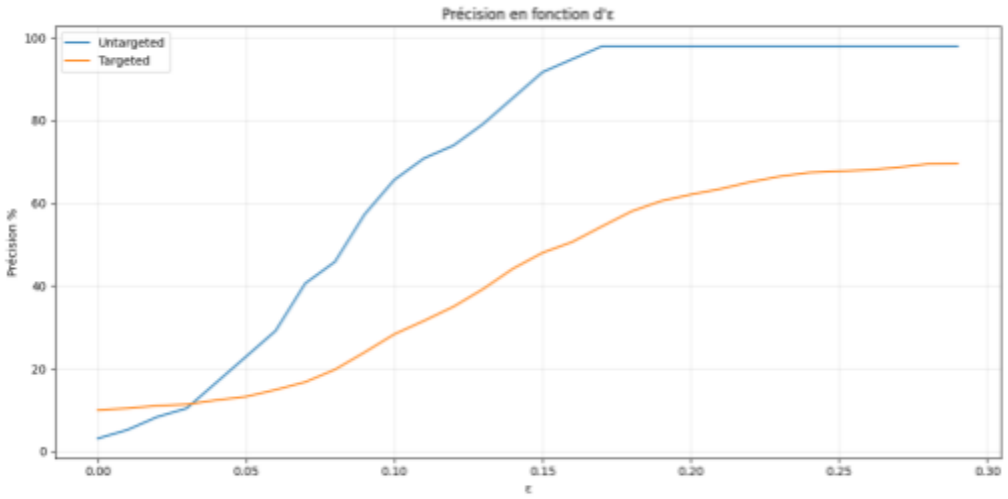


Figure 2. Graphique de la précision de FGSM en fonction ϵ sur MNIST

L'attaque *FGSM* est efficace, mais reste tout de même limitée. En effet, la qualité de l'exemple généré pour la même quantité de bruit est généralement inférieure à celle des attaques récentes. Nous pouvons d'ailleurs le voir sur le graphique plus haut, la

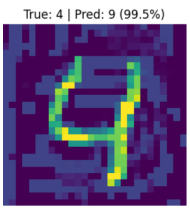
précision de la version ciblée dépasse à peine les 70% pour un ϵ réglé à 0.3. Cela est en partie explicable par le fait que la perturbation soit calculée en une seule fois.

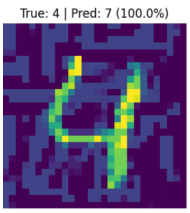
Nous allons à présent voir en quoi l'attaque *PGD* est pertinente et quelles réponses elle va proposer aux limites de *FGSM*.

b/ *PGD*

Principe

L'attaque *Projected Gradient Descent* (*PGD*) peut être vue comme une version itérative de la précédente attaque. L'idée générale est de répéter plusieurs fois *FGSM* avec une intensité très faible, en recalculant à chaque itération la direction de la perturbation à effectuer. Cette technique dispose également de deux versions. La figure de gauche montre le résultat de la version non ciblée de l'attaque, ici le modèle prédit un 9 avec 99.5% de probabilité. La figure de droite a été obtenue après une attaque *PGD*, ayant pour cible 7. La prédiction obtenue est un 7 avec une probabilité de 100%.





Implémentation

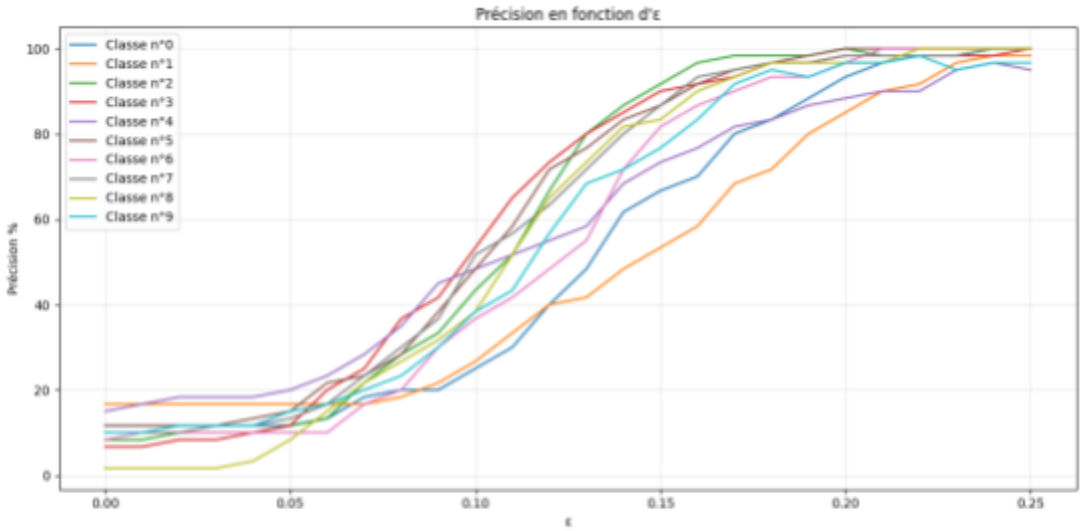
À partir de *FGSM*, l'implémentation de cette attaque ne nécessite que peu d'étapes. D'abord, nous appliquons un bruit aléatoire avant tout calcul. Cela dans le but d'éviter des minimums locaux du coût du réseau. Ensuite, nous appliquons k fois l'algorithme *FGSM* avec un facteur d'intensité α. À chaque itération, l'image est mise à jour avec la nouvelle perturbation. Ainsi, l'exemple adversarial final se construit peu à peu. Finalement, on vérifie que la modification totale ne dépasse pas un facteur ϵ.

On remarque alors qu'il y a plus de paramètres à manipuler. De manière générale, on choisi un nombre d'itérations k compris entre 20 et 100, un facteur de perturbation totale ϵ inférieur à 0.3 et une intensité de modification α approximativement égale à k/ϵ. À titre d'exemple, les images présentées ci-dessus ont été générées avec ces paramètres : (ϵ = 0.2, k = 40, α = 0.01).

Le graphique présent ci-dessous détaille, pour chaque chiffre cible, la précision de l'attaque en fonction du paramètre ϵ pour 60 images. On remarque alors que la

précision globale est nettement supérieure à celle obtenue avec *FGSM* en version ciblée. On remarque également qu'il est plus compliqué de tromper le modèle dans le sens de certains chiffres (notamment le 1 dans ce cas). Cela montre que chaque donnée n'est pas égale face à ce type d'attaque.

Cette technique propose donc de meilleurs résultats au prix de calculs plus complexes. Cependant, *PGD* reste limitée. Par exemple, trouver le bon réglage des paramètres peut nécessiter plusieurs exécutions. Aussi, le problème des minimums locaux évoqué dans la partie précédente est atténué par l'approche itérative, mais existe toujours. Une variante de *PGD* nommée *MIM* cherche à répondre à cela en ajoutant une notion d'inertie, se concentrant alors davantage sur le minimum global.



c/

Figure 3. Graphique de la précision de PGD sur MNIST en fonction ε pour chaque classe cible

DeepFool

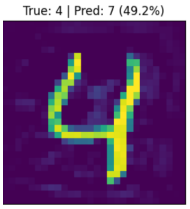
Principe

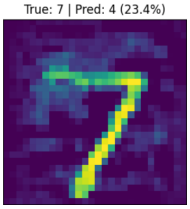
À la différence des autres attaques, *DeepFool* ne cherche pas à maximiser une valeur de coût ou à modifier les pixels selon la direction du gradient. L'idée de cette attaque est de modifier l'image de manière à ce que la prédiction franchisse une des frontières de décision du modèle. C'est-à-dire que la classification de l'image traverse une des limites internes du modèle, entraînant ainsi une erreur de prédiction.

Cette approche est particulière, car elle va calculer le bruit minimal permettant de changer la classe prédite de l'image. L'attaque fonctionne alors, par défaut, en

version non ciblée. Toutefois, il existe une variante de *DeepFool*, cherchant à franchir cette fois, non pas n'importe quelle frontière, mais plutôt la frontière avec une classe cible. Ainsi, nous sommes capables de faire fonctionner l'attaque de manière ciblée ou non ciblée.

L'image de gauche est obtenue par la version non ciblée de l'attaque. Le modèle prédit un 7 avec une probabilité de 49%. Quant à l'image de droite, elle cible le chiffre 4. Le modèle classe cette image comme étant un 4 avec une probabilité de 23%. Ces probabilités sont nettement inférieures à celles obtenues avec les précédentes attaques. Cela vient du fait que le calcul s'arrête dès que le modèle classifie le chiffre comme appartenant à une autre classe.





Implémentation

Cette attaque est plus complexe à implémenter que les dernières. En effet, elle nécessite l'accès aux frontières de décision du classifieur. La première étape de l'attaque consiste alors à linéariser le modèle, de manière à mettre en évidence ces frontières. Ensuite, tant que la classe prédite reste inchangée, on applique la perturbation minimale sur l'image, de façon à suivre le chemin le plus court vers la frontière la plus proche. Quand la classe prédite change, on continue légèrement dans cette direction afin de bien dépasser la frontière. C'est le paramètre *overshoot* qui définit l'intensité de ce dernier déplacement. Par exemple, les images ci-dessus ont été générées avec un overshoot réglé à 0.02. Il est également possible de fixer le nombre d'itérations maximal. Lors de nos tests sur 50 images, la précision de l'attaque atteint 100% au bout de 29 itérations. Le graphique présentant la précision d'attaque en fonction du nombre d'itérations est disponible en annexe.

Les résultats de cette technique peuvent paraître impressionnants, cependant, comme toute attaque adversariale, *DeepFool* présente des faiblesses.

En effet, l'attaque est moins efficace dans le cas des classifications non binaires, ou avec beaucoup de classes (Comme ici, nous avons 10 classes). C'est d'ailleurs pour cette raison qu'il nous a fallu presque 30 itérations avant d'atteindre une précision d'attaque égale à 100%.

Une deuxième limitation découle directement de la relation entre l'algorithme et le modèle. L'image adversariale obtenue fonctionne très bien pour le modèle. Cependant, si l'architecture du modèle change légèrement ou si l'on souhaite transférer l'image vers un modèle de type *black-box*, les images générées deviennent rapidement obsolètes.

d/ *UAP*

Principe

Les *Universal Adversarial Perturbations* (*UAP*) sont, comme leur nom l'indique, des perturbations universelles. Contrairement aux attaques vues précédemment, une *UAP* est générée à partir d'un jeu d'images, et peut tromper le modèle sur un grand nombre d'images de ce jeu. Le bruit n'a pas besoin d'être recalculé pour chaque image. On se contente simplement de réaliser une addition pixel par pixel, appliquant alors le bruit comme un masque sur l'image. En procédant ainsi, les *UAP* réalisent une attaque non ciblée.

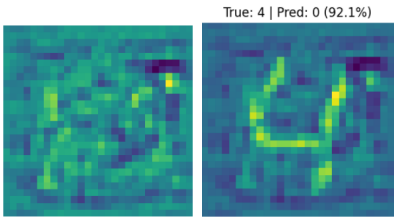
Pour générer une *UAP*, il faut itérer plusieurs fois sur le jeu de données. À chaque passage, on identifie les images qui ne trompent pas encore le modèle et on cherche une perturbation qui les amène à changer de classe. Ces modifications locales nous permettent de mettre à jour le vecteur de perturbation global.

Nous réitérons sur le jeu de données jusqu'à ce que ce vecteur de perturbation arrive à tromper le modèle sur un certain pourcentage du jeu de données, appelé *fooling rate*.

Pour calculer les perturbations locales, *UAP* repose sur un algorithme d'attaque adversariale. Il est courant d'utiliser *DeepFool*, mais nous pouvons le remplacer par d'autres attaques. Dans notre cas, nous avons essayé dans un premier temps avec *DeepFool*, puis avec *PGD*. Ces différentes approches permettent de tirer parti des avantages et inconvénients liés à chaque attaque. Nous verrons dans la partie III en quoi l'algorithme utilisé pour générer l'*UAP* amène à des résultats différents.

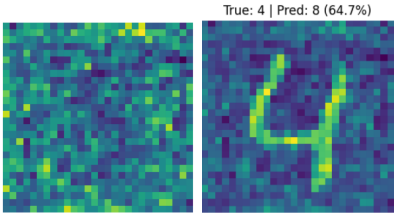
Implémentation

➔ *DeepFool*



Les figures ci-contre montrent, à gauche, une perturbation universelle générée par *UAP-DeepFool* sur 200 images du jeu de données *MNIST*. Elle a été obtenue en une seule itération et a un *fooling rate* de 87.5%. À droite, un exemple d'application sur une image du jeu de données. L'image est prédite comme étant un 0 avec une probabilité de 92%

➔ *PGD*



Pareillement, les figures ci-contre montrent, à gauche, à quoi ressemble une *UAP* générée avec *PGD*, et à droite, un exemple d'application sur une des images du *MNIST*. Comme les temps de calculs étaient sensiblement plus longs qu'avec *DeepFool*, nous avons dû baisser le nombre d'images dans le jeu de données à 100, ainsi que le *fooling rate* ciblé. Cela nous a permis d'obtenir un *fooling rate* de 62% en 2 itérations. L'image de droite est prédite comme étant un 8 avec une probabilité de 65%

Ce que l'on constate immédiatement avec les *UAP*, c'est qu'elles sont beaucoup moins discrètes que les attaques présentées précédemment. Cela découle directement du fait que l'on génère un leurre capable de tromper le modèle quand il est appliqué sur différentes images. Le coût de l'efficacité de l'*UAP* se ressent alors dans l'intensité du bruit présent. Notons qu'il est possible de régler la magnitude de l'*UAP*. C'est-à-dire qu'il est possible de limiter l'intensité des perturbations, de la même manière qu'avec le paramètre ϵ des attaques *FGSM* et *PGD*.

Un autre aspect à prendre en compte lorsque l'on travaille avec des *UAP* est le temps de génération de la perturbation. Comme nous itérons plusieurs fois sur le jeu de données complet, plus celui-ci est grand, plus les calculs sont complexes et longs.

Conclusion

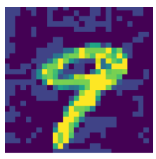
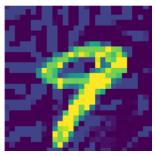
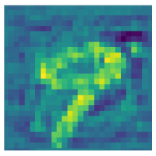
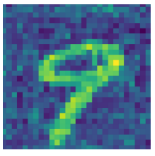
L'état de l'art intégral ainsi que l'implémentation des attaques et les ressources consultées sont disponibles en annexe.

Cette première étape nous a permis de comprendre le fonctionnement et l'ampleur de ces techniques. Lors de la rédaction de l'état de l'art, nous avons étudié cinq familles d'attaques et nous en avons retenu trois. Chacune de ces attaques a été implémentée puis testée afin de comprendre en détail son fonctionnement ainsi que l'influence des paramètres.

Dans la version complète de l'état de l'art, nous avons étudié la *Jacobian-based Saliency Map Attack* (*JSMA*), basée sur une carte de saillance. Nous n'avons pas utilisé cette attaque par la suite en raison des importants temps de calcul. C'est pour cette raison que nous avons choisi de ne pas la mentionner dans le rapport ci-présent.

Nous en avons profité de cette phase pour vérifier que les attaques testées étaient fonctionnelles et confirmer leur utilité pour la suite de ce projet : attaquer un système de détection de deepfake.

Tableau récapitulatif

Attaque	**Original**	***FGSM***	***PGD***	***DeepFool***	***UAP*-DF**	***UAP-PGD***
Temps (s)	-	10^{-3}	10^{-1}	10^{-1}	10^{3}	10^{3}
Classe prédite	9 (95%)	7 (99.8%)	7 (99.9%)	7 (46%)	0 (99.5%)	8 (99.4%)
Image						

II • Systèmes de détection de deepfakes

À présent que nous comprenons le fonctionnement et l'intérêt de ces attaques appliqués à un jeu de données simple avec un modèle peu complexe, nous allons nous pencher sur l'aspect pratique de notre travail : implémenter ces attaques sur des images de visages dans le but de tromper un modèle.

a/ Choix du modèle et du jeu de données

Nous cherchons d'abord à obtenir un modèle de classification ainsi qu'un jeu de données. En réalité, l'approche est la même que dans la première partie.

1/ Modèle

La question du choix du modèle a soulevé trois pistes principales.

Premièrement, nous nous sommes penchés sur le modèle *ResNet-50*[2]. Sorti en 2015, l'utilité première de ce modèle est la classification d'images au sens large. Il a donc fallu l'affiner avec de nouvelles données afin qu'il se comporte comme un classifieur binaire (*image vraie* ou *image fausse*). Cette première option s'est révélée intéressante, car nous avions le contrôle sur le jeu de données d'entraînement ainsi que sur les hyper-paramètres du système. Toutefois, nous avons été contraints d'abandonner cette piste, car la puissance et le temps de calcul requis n'étaient pas réalistes.

Dans un second temps, nous avons cherché un modèle déjà entraîné à classifier de manière binaire une image de visage. Nous avons choisi d'utiliser le classifieur de *selimsef*[3], vainqueur du *DeepFake Detection Challenge* (2019). Cependant, nous avons été rapidement contraints d'abandonner cette piste. En effet, il était simple de réaliser une prédiction avec ce modèle. Toutefois, la plupart des attaques nécessitant un accès aux poids du modèle, nous devions nous munir du fichier des poids ainsi que d'assez

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. arXiv. https://arxiv.org/abs/1512.03385

[3] Selimsef. (2020). Selimsef/dfdc_deepfake_challenge: A prize winning solution for DFDC Challenge. GitHub. https://github.com/selimsef/dfdc_deepfake_challenge

de puissance de calcul afin d'obtenir la meilleure perturbation. Or, le modèle ayant été créé 6 ans auparavant, les liens vers les poids de ce dernier étaient obsolètes. Nous avons donc eu des difficultés à obtenir les paramètres du classifieur. De plus, le poids du modèle s'est révélé être trop important par rapport à la puissance dont nous disposions.

Finalement, après ces deux abandons, nous avons utilisé le classifieur de *i3p9*[4], une version affinée de Xception[5], un modèle entraîné avec la librairie *torch*. Ce modèle présente les avantages d'être facilement accessible et léger. Nous avions donc la possibilité de faire des prédictions ainsi que d'accéder aux gradients du modèle.

Comme précisé dans la première partie, nous travaillons avec le module *Tensorflow*. Par souci de compatibilité, nous avons donc converti le modèle en un format utilisable avec *Tensorflow* à l'aide d'outils tels que *nobuco*.

2/ Choix du dataset

En 2019, les chercheurs de l'université de Munich publient le jeu de données *FaceForensics++*[6]. Il est construit à partir de mille vidéos issues de Youtube sur lesquelles une transformation a été appliquée. Nous avons donc téléchargé puis extrait vers des images la moitiée de ces vidéos, portant la taille de notre jeu de données au-delà de 2 Téraoctets.

Une fois le jeu de données téléchargé, nous avons cherché à vérifier que les prédictions du modèle sur ce dernier étaient cohérentes.

b/ Évaluation du modèle

Cette étape d'évaluation est cruciale. En effet, on ne veut pas construire une attaque sur le modèle en utilisant une image qu'il n'est pas capable de reconnaître.

[4] I3p9. (2021). GitHub - i3p9/deepfake-detection-with-xception : Deepfake detection. GitHub. https://github.com/i3p9/deepfake-detection-with-xception
[5] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. arXiv. https://arxiv.org/abs/1610.02357
[6] Rössler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., & Nießner, M. (2019). FaceForensics++: Learning to detect manipulated facial images. arXiv. https://arxiv.org/abs/1901.08971

Nous avons donc vérifié que les prédictions du modèle correspondent bien aux étiquettes des images.

1/ Premiers résultats

Il a d'abord été nécessaire d'extraire les visages des images. Pour cela, nous avons utilisé l'outil de *freearhey*[7] qui découpe les visages présents dans une image, créant ainsi autant de sous-images que de visages. Nous avons extrait 4356 images de visage. Comme nous pouvons le voir sur la figure ci-dessous, les résultats obtenus sont mauvais. En effet, les distributions des prédictions des images vraies et fausses sont presque superposées. La précision (proportion des cas réellement faux parmi tous les cas classés faux) du modèle et le recall (proportion des cas faux classés faux) sont alors restés aux alentours de *0.50*.

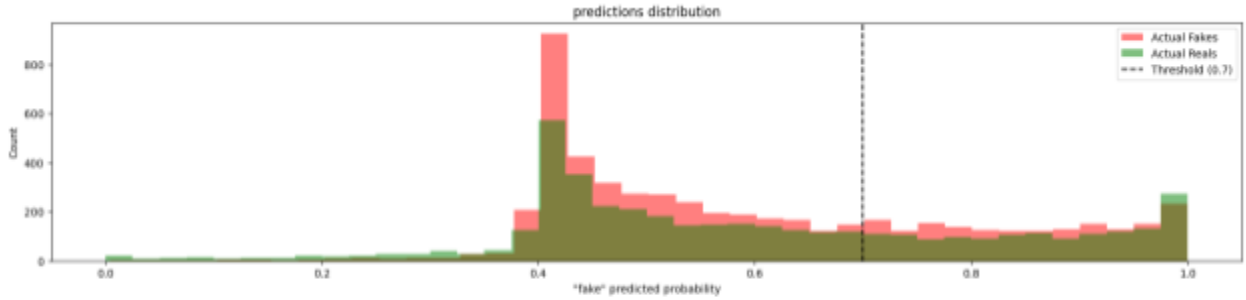


Figure 4. Distributions des prédictions du modèle avant tamisage du jeu de données

Nous avons donc cherché à afficher les données les moins bien prédites : des visages difficilement perceptibles, avec un objet bloquant une partie du visage. Parfois même, des formes étaient interprétées comme étant un visage (des exemples de photo ainsi que les métriques de cette étape sont disponibles en annexe). Nous avons donc cherché à corriger cela.

2/ Amélioration du dataset

Nous avons alors deux possibilités qui s'offrent à nous pour résoudre ce problème. La première est d'affiner le modèle. La seconde est d'adapter le jeu de données. Nous sommes partis sur la deuxième solution pour des raisons de puissance de calcul et de faisabilité.

[7] Freearhey. (s. d.). GitHub - freearhey/face-extractor : Python script that detects faces on the image or video, extracts them and saves to the specified folder. GitHub. https://github.com/freearhey/face-extractor

Nous avons alors utilisé l'outil de *dullage*[8], visant à aligner les yeux avec l'horizon ainsi qu'à centrer le visage dans l'image. Ainsi, les photos avec au moins un œil caché étaient filtrées. Le fait que les visages soient centrés et à la même échelle a joué un rôle important dans l'amélioration des résultats. Ensuite, nous avons remarqué que certaines images étaient zoomées ou pixelisées. Pour corriger cela, nous avons fixé un seuil de qualité d'image minimale afin de filtrer de manière plus fine.

3/ Résultats après traitement des images en entrée

Cette étape d'écrémage du jeu de données s'est révélée cruciale.

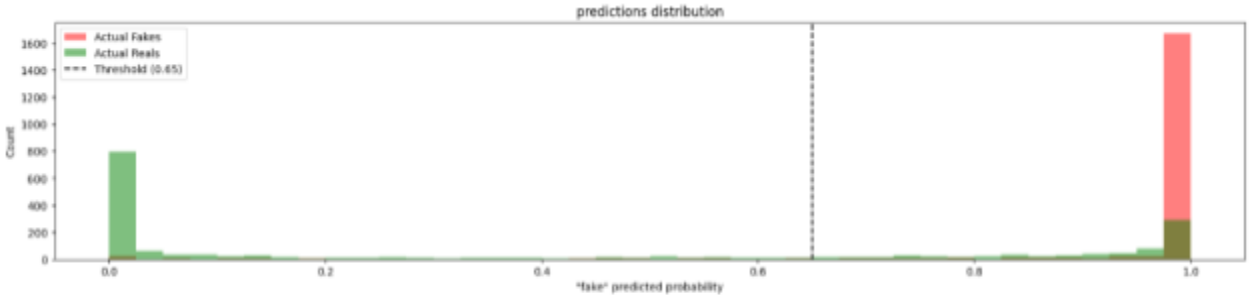


Figure 5. Distributions des prédictions du modèle après tamisage du jeu de données

Comme nous le remarquons sur la figure ci-dessus, les résultats d'évaluation du classifieur sur le jeu de données filtré sont conformes à nos attentes. En effet, on observe un clair divorce entre les deux distributions de prédictions. La précision atteinte est de 0.96 pour la classe *vraie* (en vert) et 0.73 pour la classe *fausse* (en rouge). Ce qui est intéressant, c'est le recall de la classe *fausse*. Il est égal à 0.97. Cela signifie que seulement 3% des images de deepfakes seront qualifiées comme étant vraies. Ce score de recall est bien supérieur à celui de la classe *vraie* (0.64). Nous avons obtenu ces métriques en fixant un seuil de décision à 0.65. L'objectif était d'avoir le moins de faux négatifs.

Cet ensemble de données va servir de base pour la suite : attaquer directement le modèle de détection de deepfake.

[8] dullage. (2025). GitHub - dullage/eyelign: A tool to align multiple portrait photos by eye position. GitHub. https://github.com/dullage/eyelign

III• Application aux deepfakes

Une fois notre système de détection mis en place, nous avons pu mettre en application les attaques adversariales vues en première partie afin de le tromper.

Pour cela, nous avons ajusté les implémentations faites sur le jeu de données *MNIST* de manière à ce que les données concordent avec la dimension d'entrée du classifieur. Cet ajustement se réalise en deux temps. D'abord, nous modifions la structure de l'attaque. De manière à traiter des images qui ont trois canaux. Puis, nous appliquons à chaque image un prétraitement qui consiste, entre autres, à changer l'ordre des canaux de couleurs, redimensionner puis normaliser l'image.

La dimension des images que nous traitons est de 299 par 299. Cela signifie que pour chaque image, il y a environ 350 fois plus de paramètres à traiter que dans le cas des images du *MNIST* (qui sont de dimension 28 par 28). Ainsi, les temps de calculs des attaques se sont substantiellement accrus.

Pour cette raison, en plus de *Deepnote*, nous avons mis en place une instance *EC2* ainsi qu'un *bucket* hébergés chez AWS, de manière à stocker et manipuler l'ensemble des données. Nous avons choisi d'utiliser ces technologies pour répondre au problème de la multiplication des temps de calculs. Ainsi, nous étions capables de réaliser et d'exécuter des algorithmes sophistiqués et coûteux en ressources.

a/ *PGD*

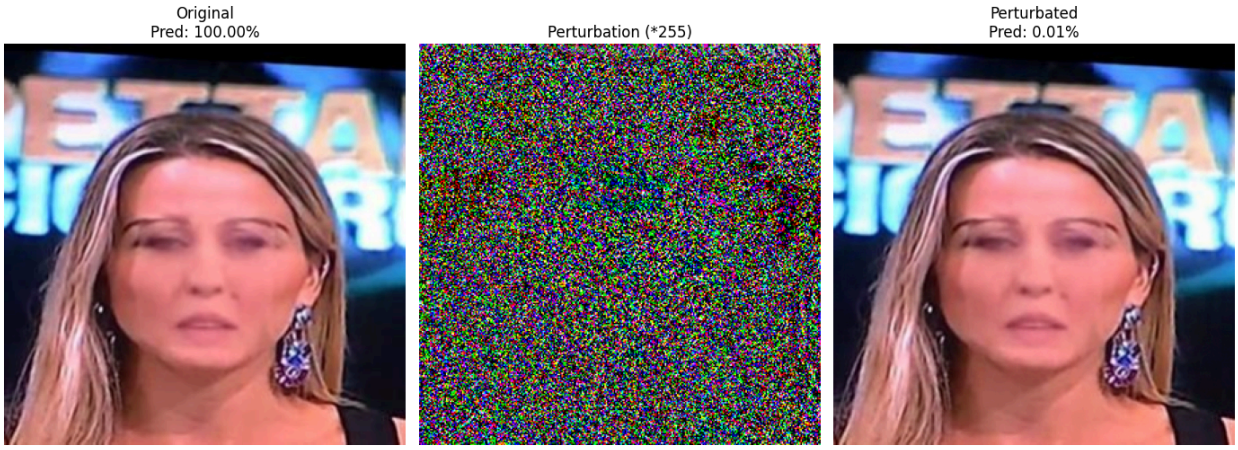
Application

Adapter *PGD* à de grandes images disposant de trois canaux s'est révélé être une tâche simple. L'utilisation de la librairie *numpy* nous a facilité la modification de l'attaque. La plupart des calculs effectués sur les images du *MNIST* sont restés valables. Nous avons choisi d'intégrer uniquement la version non ciblée de l'attaque. Ce choix découle directement de la classification binaire.

Comme prévu, le temps de calcul s'est accru, passant d'environ 0.2s (pour une image du *MNIST*) à 0.7s. Notons que le réglage des paramètres utilisés est nettement différent dans notre cas. Cela explique pourquoi l'augmentation du temps de calcul est peu significative.

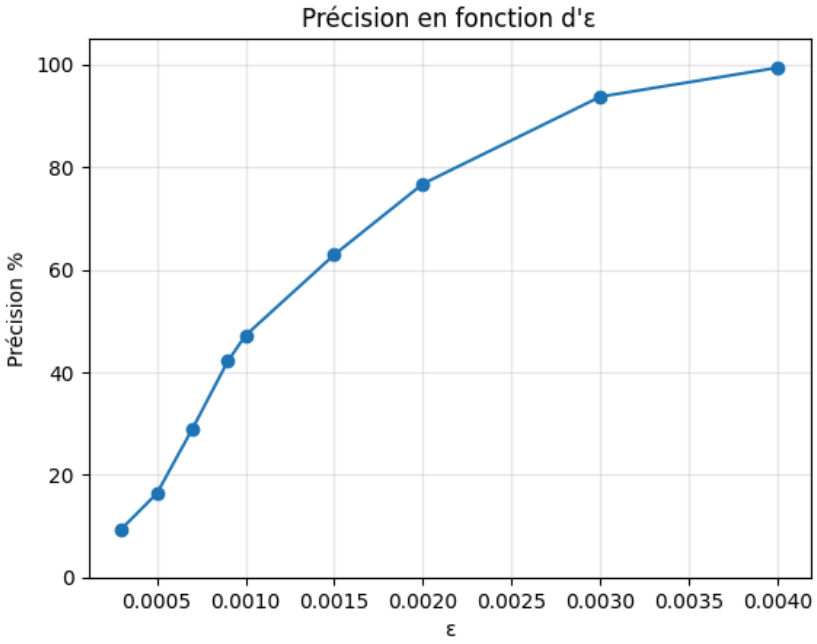
Résultats

Les paramètres de l'attaque *PGD* restent les mêmes, à savoir : ϵ (perturbation totale), k (nombre d'itérations) α (perturbation à chaque itération).

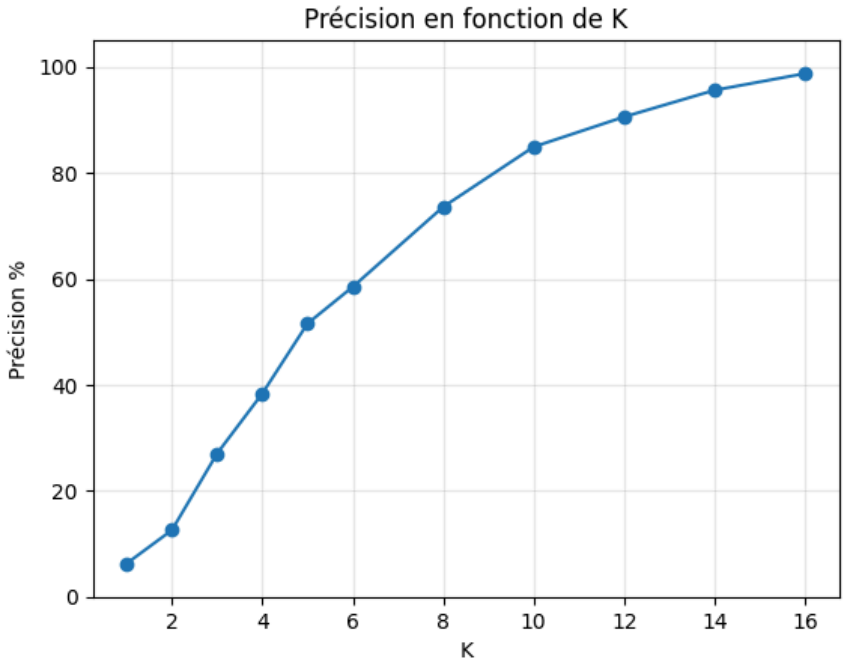


La figure ci-dessus montre à gauche, une image de deepfake issue du jeu de données. Notre modèle prédit que c'est un deepfake à 100%. Au milieu, il y a la perturbation calculée par *PGD*. Nous l'avons multipliée par 255 de manière à mieux la visualiser. Finalement, l'image de droite correspond à l'original après application de la perturbation. Le modèle prédit l'image perturbée comme étant vraie à 99.9%. Nous pouvons donc dire que l'attaque a été réussie. Les paramètres utilisés sont les suivants : $\epsilon = 10^{-2}$, $k = 5$ et $\alpha = 10^{-3}$.

Les facteurs d'intensité de perturbation (ϵ et α) sont bien plus faibles que lors de l'attaque sur les données du *MNIST*. Cela découle directement du fait que l'image est plus étendue. Ainsi, la proportion de l'image à modifier est bien inférieure à celle d'une image provenant du *MNIST* pour un succès d'attaque similaire. Le graphique



présenté ci-dessus illustre bien l'influence de l'amplitude de la perturbation totale sur le succès de l'attaque. Il a été généré à partir d'une trentaine d'images pour lesquelles le nombre d'itérations était fixé à 5. On remarque que même pour une petite perturbation, l'attaque reste très efficace.



Le graphique ci-contre a été généré de la même manière que le précédent. Il présente l'efficacité de l'attaque en fonction du nombre d'itérations quand l'amplitude de la perturbation totale (ϵ) est réglée à 5×10^{-3}. Sans surprise, nous remarquons que le nombre d'itérations est fortement lié au succès de l'attaque.

D'une manière générale, il est essentiel d'appréhender la manière dont les différents paramètres influent sur l'efficacité globale de l'attaque ainsi que sur la discrétion de la perturbation générée. Nous le verrons par la suite, cette compréhension s'est avérée particulièrement utile quand nous avons intégré *PGD* dans le but de générer des *UAP*.

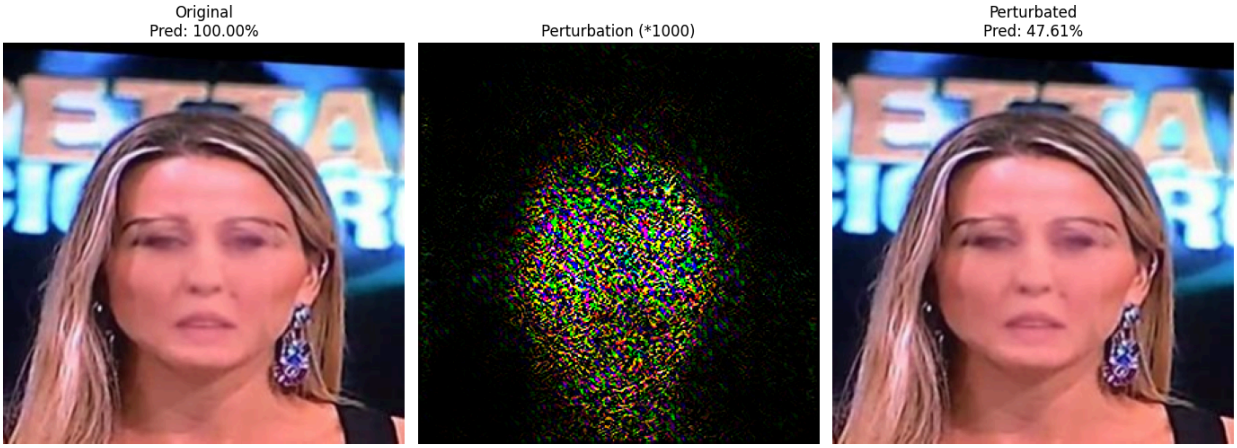
b/ *DeepFool*

Application

De la même manière que pour *PGD*, *DeepFool* a été intégrée sans entrave particulière. Nous avons choisi d'intégrer la version par défaut de cette attaque, à savoir, la version non ciblée. Le fonctionnement particulier de *DeepFool* ne nécessite que très peu de paramètres, leur réglage a donc été rapide. On retrouve un nombre d'itérations maximum ainsi qu'une valeur d'*overshoot* qui, rappelons-le, spécifie la marge avec laquelle la frontière doit être dépassée.

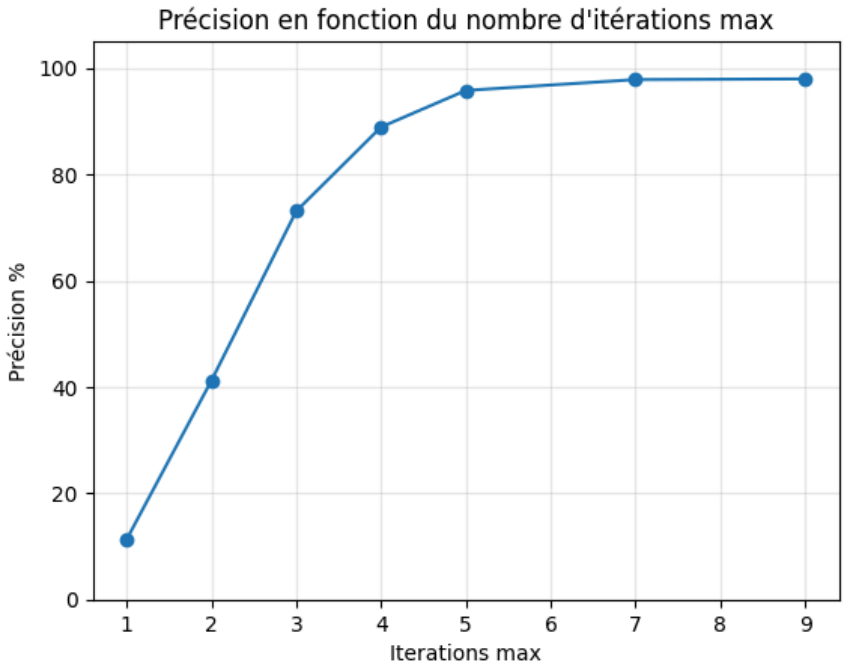
Le temps d'exécution passe alors de 0.5s (attaque sur une image du *MNIST*) à 4s. Cette différence est plus importante que dans le cas de *PGD*. Cela découle du fait que les réglages de l'attaque n'ont pas été modifiés.

Résultats



Sur la figure ci-dessus, nous avons, à gauche, une image de deepfake non modifiée issue du jeu de données. Elle est prédite comme étant un deepfake avec une probabilité de 100%. La perturbation calculée par *DeepFool* est au milieu. Nous l'avons multiplié par un facteur 1000 afin de la rendre visible. Finalement, à droite, nous avons l'image perturbée par *DeepFool*. Elle est prédite comme étant réelle avec une probabilité d'environ 52%. L'attaque fonctionne bien, le modèle a été leurré avec succès. L'attaque a été réalisée avec un paramètre *overshoot* réglé à 0.02 et un nombre d'itérations maximum fixé à 10. Ce paramètre visant à limiter le nombre d'itérations influe directement sur la réussite de l'attaque. En effet, comme présenté dans le graphique ci-dessous, la précision du modèle évolue rapidement puis sature après la cinquième itération. Maîtriser ce paramètre, c'est limiter le temps de calcul relatif à

l'attaque en réduisant légèrement son efficacité.



Comparer l'efficacité d'une attaque passe par deux points essentiels. Le premier consiste à vérifier si le classifieur est correctement trompé. Dans notre cas, nous pouvons comparer en utilisant la prédiction du modèle. Le second concerne la qualité de la perturbation. En clair, "Est-ce que l'image générée est assez discrète pour passer inaperçue à l'œil humain ?". Cette comparaison peut être réalisée de manière précise, en utilisant le score MS-SSIM. Si le score de deux images est proche de 1, c'est qu'elles sont très similaires. Au contraire, un score avoisinant 0 sera retourné si les images sont radicalement différentes. Nous utilisons donc ce score pour comparer la discrétion des attaques entre elles. Ainsi, dans le cas de la figure présentée dans la partie *PGD*, le score de similarité entre les deux images est de 99.966%. Dans le cas de *DeepFool*, ce score monte à 99.998%. Cela découle directement de la manière d'opérer de l'attaque : calculer la perturbation minimale, suffisante pour changer la classe prédite.

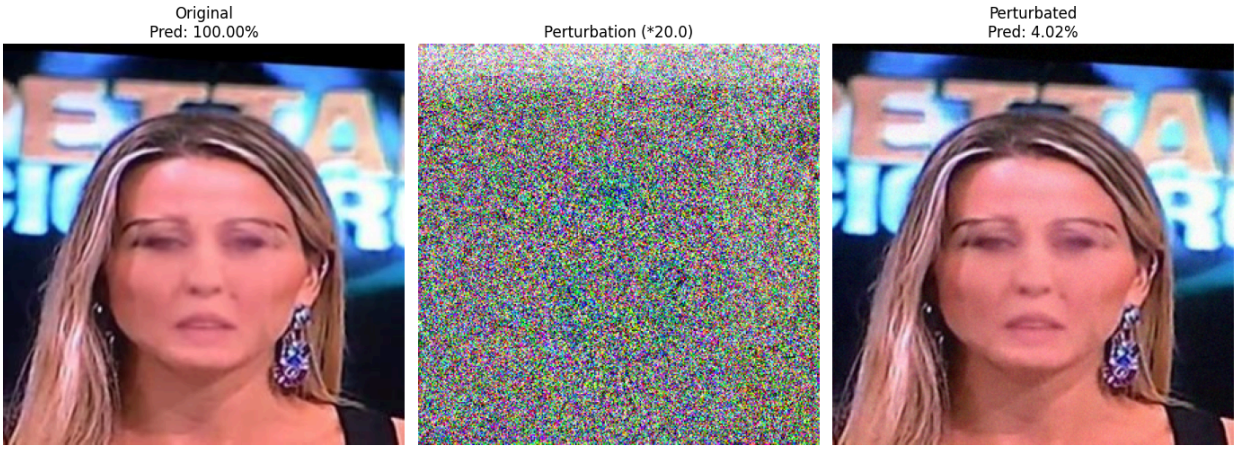
c/ *UAP*

UAP étant un algorithme qui se repose sur des attaques adversariales telles que *DeepFool* ou *PGD*, nous l'avons adapté facilement en utilisant les nouvelles versions des attaques qu'il intègre.

Le nombre d'images utilisées pour générer l'*UAP* en utilisant les algorithmes *PGD* et *DeepFool* s'est rapidement retrouvé limité. En effet, pour une perturbation universelle basée sur 784 images, le temps de calcul s'élevait à plusieurs heures.

1- *PGD*

Nous avons généré plusieurs perturbations universelles avec plusieurs paramètres. Finalement, l'*UAP* qui s'est révélé être la plus efficace a été générée avec les paramètres de *PGD* réglés de cette manière : $\epsilon = 0.02$, $\alpha = 10^{-3}$ et $K = 100$.



La figure ci-dessus montre les résultats de l'attaque. À partir d'une image originale (à gauche) prédite comme étant un deepfake à 100%, nous appliquons la perturbation universelle calculée au préalable (image du milieu, nous l'avons augmenté afin de mieux la visualiser). Finalement, nous obtenons l'image présente à droite, prédite comme étant réelle avec une probabilité de 96%. Nous pouvons alors dire que l'attaque fonctionne dans ce cas. Nous avons ensuite calculé le *fooling rate* sur un échantillon de 1632. C'est de cette manière que nous avons obtenu un score égal à 97.24%.

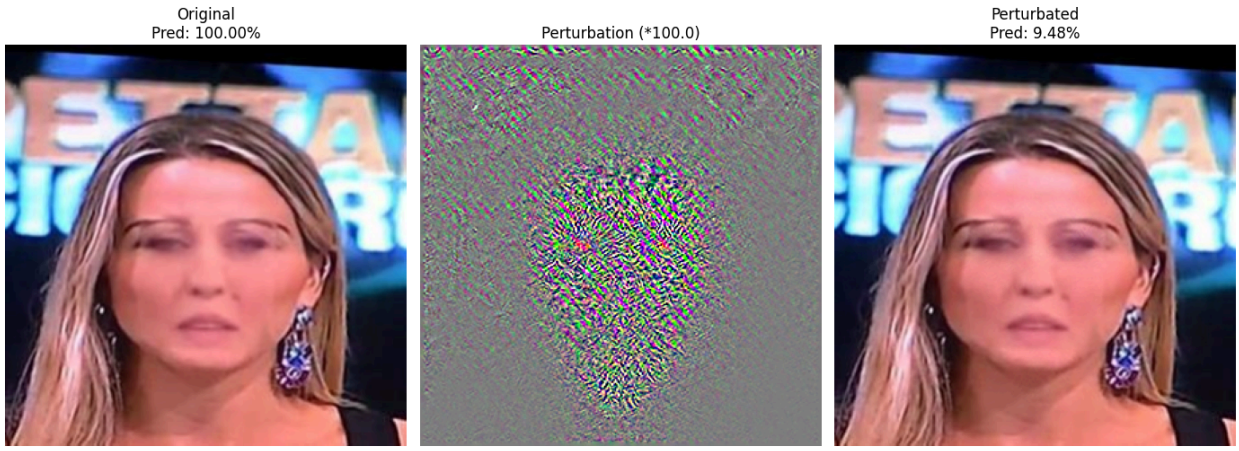
Le point faible général des attaques universelles est que l'exemple adversarial est peu similaire à l'image originale. Dans notre cas, le score SSIM moyen est de 96.44%, avec un minimum de 92.887%, ce qui est légèrement plus faible que les

attaques individuelles mais reste raisonnablement peu visible. À titre d'exemple, le score de similarité entre les deux images ci-dessus est de 98.147%.

En regardant bien la perturbation générée, nous remarquons la présence d'une bande claire en haut de l'image. Celle-ci est causée par le prétraitement de notre jeu de données : lorsque le visage est trop proche de la bordure haute de l'image d'origine, le recentrage effectué pour améliorer les performances du modèle entraîne l'apparition d'une bande noire.

2- *DeepFool*

Tout comme pour *PGD*, nous avons testé plusieurs paramètres pour générer l'*UAP*. *DeepFool* possédant moins de paramètres, cette étape a été plus rapide. Nous avons atteint le *fooling rate* cible de 90% pour l'ensemble des 784 images sur lesquelles nous avons généré la perturbation. Pour cela, le paramètre *overshoot* était réglé à 0.3, dans le but d'obtenir une *UAP* générale, fonctionnant le mieux possible sur des images absentes du support de calcul.



La figure ci-dessus montre l'application de la perturbation calculée au préalable sur une image classifiée comme étant un deepfake à 100% (image de gauche). La perturbation (image du milieu) a été augmentée cinq fois plus que celle présentée dans la partie précédente. Le score SSIM de ces images est de 99.941%. Cette forte similarité s'explique par le fonctionnement de l'algorithme de *DeepFool* qui, rappelons-le, cherche à chaque itération la perturbation minimale pour franchir la frontière de décision. Le leurre généré est alors très spécifique aux images support.

De la même manière qu'avec la génération de l'*UAP* en utilisant *PGD*, nous avons évalué cette perturbation sur un ensemble de 1632 images. Nous avons obtenu un *fooling rate* égal à 82.97%. Cela signifie que dans 17% des cas, l'*UAP* a échoué à faire changer la classe prédite par le modèle. On comprend alors que la perturbation est certes plus discrète que celle générée dans la partie précédente (on parle d'un score de similarité moyen supérieur à 99%). Toutefois, son efficacité reste limitée.

En effet, nous remarquons que la perturbation générée se concentre davantage sur la zone du visage, qui est plus à même de changer. Nous sommes d'ailleurs capables de distinguer la zone des yeux.

Notons que sans l'étape cruciale d'alignement des visages en fonction des yeux, la perturbation universelle obtenue n'aurait pas été aussi efficace.

3- Essais en *black-box*

L'une des particularités des *UAP*, telles qu'elles sont décrites par Moosavi-Dezfooli, Fawzi, Fawzi et Frossard[9], est qu'elles conservent une bonne efficacité même en changeant de modèle. Cela les rend intéressantes pour des attaques en "*black-box*", c'est-à-dire sans accès au fonctionnement interne du modèle.

C'est dans cette optique que, d'après l'idée de nos encadrants, nous avons essayé de tromper un modèle de détection de deepfakes disponible en ligne. Pour cela, nous avons appliqué les *UAP* générées à partir de notre modèle sur des images de deepfakes générées par un outil en ligne. Les tests qui suivent ont été réalisés afin de valider cette propriété dans un cadre réaliste et de tester l'efficacité de nos perturbations sur un autre système de détection.

Nous nous sommes donc dans un premier temps servi de l'outil aifaceswap[10] pour générer des deepfakes en faisant un face swap des visages de célébrités. Dans la figure ci-dessous, un exemple avec Céline Dion et Donald Trump.

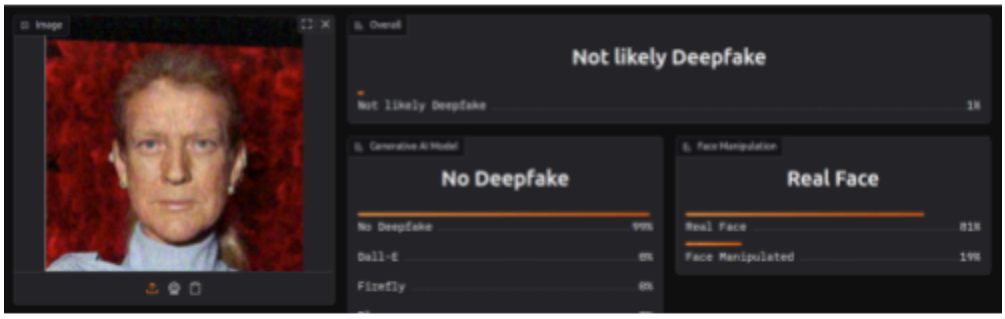
[9] Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., & Frossard, P. (2016). Universal adversarial perturbations. arXiv. https://arxiv.org/abs/1610.08401
[10] Swap face with Ai Online. AI Face Swap. (n.d.). https://aifaceswap.io/

Figure 6. Face swapping entre Céline Dion et Donald Trump

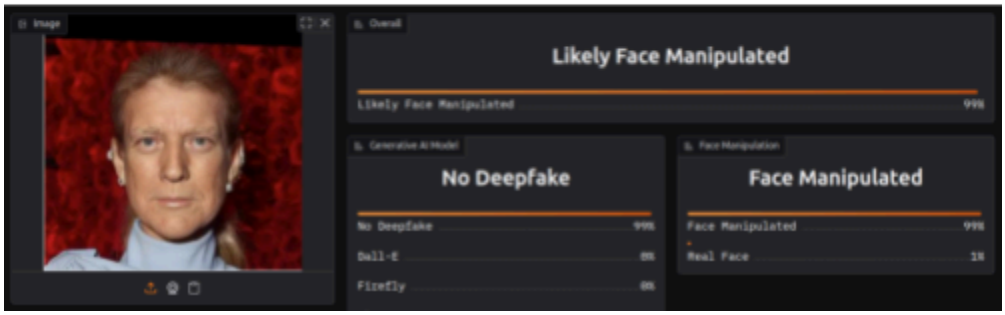
Nous avons ensuite testé nos attaques adversariales sur le modèle de détection de deepfake proposé par faceonlive.[11] Pour cela, nous avons appliqué le même prétraitement que pour les images de notre précédent jeu de données.

Deepfake seul :

Avec la perturbation universelle PGD :



Avec la perturbation universelle DeepFool :



[11] FaceOnLive. (n.d.). Deepfake Detector - a Hugging Face Space by FaceOnLive. https://huggingface.co/spaces/FaceOnLive/Deepfake-Detector

Comme nous pouvons le constater, nous sommes capables de tromper le système, cela en appliquant sur l'image la perturbation universelle précédemment générée avec *PGD*.

En revanche, nos essais avec la perturbation universelle générée par *DeepFool* se sont révélés moins concluants, n'affectant presque pas la prédiction du système.

Ces résultats viennent corroborer l'hypothèse selon laquelle *UAP-PGD* est plus facilement transférable que *UAP-DeepFool*, le bruit de ce premier étant moins spécifique aux images sur lesquelles il a été calculé. Aussi, la perturbation obtenue par *PGD* cherche à radicalement faire changer la prédiction du modèle. Tandis que celle calculée par *DeepFool* a pour objectif d'être la plus fine possible, juste assez pour changer la classe prédite.

Conclusion

Au cours de ce projet, nous avons, dans un premier temps, étudié différents types d'attaques adversariales et leur fonctionnement, puis nous les avons appliquées à un modèle de détection de deepfake par face swapping. L'objectif était d'évaluer dans quelle mesure ces attaques pouvaient dégrader les performances du détecteur sans altérer visiblement les images.

Nous avons été surpris par l'efficacité de ces attaques. Certaines se sont révélées très performantes, avec des perturbations quasiment imperceptibles pour l'œil humain. Cela met en évidence que les systèmes de détection de deepfakes, même s'ils constituent une première barrière contre les menaces qu'ils représentent, restent vulnérables.

Des améliorations restent encore à apporter vis-à-vis du travail fait : les contraintes temporelles du projet et la puissance de calcul de notre matériel ont limité nos essais, et nous aurions aimé pouvoir expérimenter l'efficacité de ces attaques sur des modèles de détection de deepfake modernes, avec des jeux de données plus récents.

Au-delà de ces perspectives d'amélioration, une piste de poursuite du projet se dessine : l'évaluation des stratégies de défenses contre ces attaques. En effet, maintenant que nous avons mis en lumière les vulnérabilités des systèmes de détection de deepfakes, il nous semble légitime de réfléchir à comment les renforcer.

Un exemple de technique de défense consiste à ajouter des images perturbées dans le jeu de données servant à l'entraînement du classifieur. On parle alors d'entraînement adversarial. D'autres approches se concentrent plutôt sur la détection préalable des perturbations, avant l'évaluation par le modèle.

La mise en place de telles mesures est indispensable dans un contexte où les attaques se renouvellent et se complexifient constamment. La défense ne peut se contenter de simplement réagir : elle doit faire preuve d'anticipation et d'innovation. L'enjeu final serait de conserver une robustesse afin que les systèmes de détection aient toujours une longueur d'avance sur les méthodes de contournements récentes.

Finalement, ce projet en collaboration avec le laboratoire du *GREYC* s'est révélé très enrichissant, et nous a permis de renforcer nos compétences à la fois techniques et méthodologiques.

D'abord, en intelligence artificielle, par la maîtrise de techniques dites "*white-box*". Nous avons également affermi notre compréhension des chaînes d'attaque mises en place par des acteurs malveillants. La première partie nous a permis de perfectionner nos compétences rédactionnelles ainsi que notre esprit critique. En outre, les défis relevés par le manque de puissance de calcul nous ont permis d'affiner nos connaissances en infonuagique.

Nous vous remercions de votre lecture.