PFE 1846

DeepTruth

ECE PARIS
ÉCOLE D'INGÉNIEURS

Toquebiau Maxime, Pécresse Clément, Thery Antoine,
Guillermou Tom, Prat Arthur, Coste Corentin

Abstract :

Dernière technologie de la désinformation, les *deepfakes* sont des vidéos comportant des manipulations faciales permettant de faire dire n'importe quoi à n'importe qui. Face à ce problème, chercheurs et grands groupes se mobilisent pour informer et combattre les *deepfakes* sur leurs plateformes. Notre projet DeepTruth s'inscrit dans ce combat. DeepTruth est une extension web permettant de détecter la présence d'un *deepfake* dans n'importe quelle vidéo sur internet. Grâce à des modèles complexes de *deep learning* et une utilisation judicieuse du *fingerprinting* de vidéos, nous proposons une analyse rapide et fiable. De plus, notre approche d'une extension de navigateur nous permet d'offrir ce service sur la totalité du web, et pour tous ses utilisateurs.

Sommaire

Introduction

L'émergence de la désinformation sur le web est devenue un problème majeur. Avec les *fake news*, certains acteurs politiques ou commerciaux sont capables d'influencer massivement les populations à travers des articles disponibles en ligne. Depuis peu, on voit apparaître des techniques basées sur l'Intelligence Artificielle (IA) pour générer du contenu fallacieux. Il est maintenant possible de créer des *fake news* en utilisant des algorithmes de *deep learning*, et certains sites se font déjà la vitrine de ces techniques [1][2]. Un domaine de la recherche se penche également sur la génération de vidéos truquées. Le terme "***deepfake***" caractérise des vidéos synthétisées automatiquement par ordinateur montrant des personnes dont le visage a été remplacé par celui d'une autre, ou dont les mouvements faciaux ont été modifiés pour prononcer un autre message. Elles sont utilisées pour générer des *fake news*, des vidéos de politiciens tenant des propos compromettants [3] et même du contenu pornographique mettant en scène des célébrités [4].

Cette multiplication des contenus fallacieux est rendue possible par un manque de remise en question du contenu web par la plus grande majorité des utilisateurs, notamment sur les réseaux sociaux. Or ce constat est tout sauf surprenant. Nous sommes habitués à croire les propos des médias en général. De plus, très peu de gens savent comment déceler les contenus mensongers. De ce constat émerge notre problématique : **Comment permettre aux utilisateurs du web d'authentifier le contenu qu'ils visionnent ?** Pour répondre à ce problème, différentes solutions sont envisageables. Plusieurs sites ont déjà banni les *deepfakes* de leur plateforme. Twitter a récemment proposé une politique de détection et d'alerte des vidéos truquées [5]. De son côté, Facebook a lancé une campagne de lutte contre les *deepfakes* via la création d'un dataset de vidéos truquées et le développement d'algorithmes de détection [6]. Le danger est donc clairement identifié par les grands acteurs du web. Mais malgré leur vaste suprématie, leurs plateformes ne représentent pas la totalité de l'Internet. Il faut donc trouver des outils pour lutter contre les *deepfakes* sur les réseaux sociaux et ailleurs.

C'est donc ici que rentre en jeu notre Projet de Fin d'Etudes (PFE). **DeepTruth** est une extension web permettant de détecter les *deepfakes* présents dans le contenu des pages web. Contrairement aux géants du web précédemment cités, qui gèreront le problème seulement sur leurs plateformes, notre extension aura l'avantage d'être utilisable sur n'importe quel site. Cette approche n'est pas nouvelle dans le combat contre la désinformation. En témoigne plusieurs extensions web permettant de détecter les *fake news* [7]. DeepTruth analysera donc toute vidéo présente dans les pages visitées pour alerter ou non l'utilisateur de la présence d'un *deepfake*. Pour répondre de la meilleure des manières au problème initial, nos mots d'ordre seront fiabilité, rapidité et universalité.

Ce projet est réalisé en partenariat avec **Nextino**. Nextino est un centre de recherche en IA basé à Orléans. Né au sein du groupe de sécurité informatique français Atempo-Wooxo, Nextino baigne dans un écosystème orienté sur les données, leur protection et leur utilisation. Nous leur avons proposé ce projet dans l'espoir de pouvoir profiter de leur expertise et de leurs ressources. Nous reviendrons donc dans ce rapport sur les contributions de Nextino à notre projet, ainsi que sur les différentes parties qui leurs seront utiles dans le futur.

Le travail sur ce projet a été séparé en trois axes : la détection de *deepfakes*, le *fingerprinting* de vidéos et l'extension web. Dans ce rapport nous aborderons donc ces trois axes en détaillant pour chacun d'entre eux l'état de l'art, les techniques utilisées et les résultats obtenus. Nous expliquerons ensuite le fonctionnement de DeepTruth dans son intégralité, pour finir sur les suites possibles du développement de ce projet.

1. Détection de *deepfakes*

1.1. Etat de l'art

Le coeur de notre projet réside dans notre capacité à trouver un *deepfake* dans une vidéo. Pour ce faire, nous allons devoir définir une stratégie d'analyse de cette vidéo. Nous sommes donc dans le domaine de la *Computer Vision*, qui regroupe toutes les techniques d'analyse d'images. Ce domaine de l'informatique est extrêmement développé : il existe une multitude de techniques d'analyse automatique d'images. Depuis une vingtaine d'années, le *deep learning* (ou "apprentissage profond") a fait son apparition dans l'analyse d'images avec comme fondement les *Convolutional Neural Networks* (CNN, ou "réseaux de neurones à convolution"). Cette architecture reprend le concept de réseau de neurones artificiels et l'applique à la *Computer Vision* pour tenter d'extraire automatiquement des informations d'une image. Rapidement les CNN se sont montrés extrêmement efficaces pour des tâches de classification d'image, de détection d'objets ou encore, plus récemment, de génération d'image. Il n'y a aujourd'hui aucune autre technique qui rivalise avec les CNN pour analyser des images automatiquement, pour peu qu'on ait la puissance de calcul nécessaire à disposition. C'est donc naturellement vers ces techniques que nous nous sommes tournés pour répondre à notre problème.

Notre étude de l'état de l'art s'est basée sur le papier scientifique *"**FaceForensics++**: Learning to Detect Manipulated Facial Images"* par Rössler et al. [8]. Publié en août 2019, ce papier propose un rapide aperçu des différentes techniques de manipulation faciale, un dataset d'images manipulées ainsi qu'un benchmark des différentes techniques de détection de manipulation. Le dataset contient 1000 vidéos manipulées grâce à différentes techniques : *Face2Face* [9], *FaceSwap* [10], *DeepFakes* [11] et *NeuralTextures* [12]; ainsi que leur version d'origine. Elles ont toutes été prises de Youtube et montrent toutes une personne parlant face à la caméra. Plusieurs niveaux de compression sont disponibles pour correspondre au cas réel des vidéos postées sur les réseaux sociaux. Vous trouverez en annexe 1 une rapide étude des techniques de manipulation précédemment citées.

Grâce au riche dataset de FaceForencics++ [8], un benchmark de l'état de l'art est réalisable. Les auteurs proposent donc une étude des différentes techniques de détection de manipulation faciale et évaluent leurs performances grâce à un test automatisé. Voilà un aperçu des techniques étudiées :

- Fridrich et al. [13] présentent une technique utilisant des filtres paramétrés manuellement pour détecter des manipulations dans une image. Le résultat de 162 de ces filtres créent le vecteur de *features* ("caractéristiques") dont ils se servent pour entraîner un classifieur de type *Support Vector Machine* (SVM). C'est ce modèle qui déterminera, à partir des *features*, la validité de l'image. Ce modèle a remporté le premier IEEE Image Forensics Challenge [14] en 2014.
- Cozzolino et al. [15] réutilisent les filtres créés manuellement par Fridrich et al. [13] et les intègrent à une architecture de CNN.
- Bayar et Stamm [16] utilisent une architecture contrainte de CNN qui vise à extraire les *features* bas niveau de l'image.
- Rahmouni et al. [17] proposent également leur propre architecture de CNN pour détecter des parties d'images générées par ordinateur.
- Afchar et al. [18] présentent leur architecture de CNN MesoInception-4, inspiré du célèbre InceptionNet [19], pour détecter les manipulations faciales. L'architecture mélange des modules d'inception à des modules de convolution classique.

- Enfin, les auteurs proposent leur propre solution qui utilise l'architecture **Xception** [20]. Également inspiré de InceptionNet [19], cette architecture a fait ses preuves par le passé sur des tâches de classification d'images. Ici les auteurs font du *transfer learning* en remplaçant la dernière couche dense du réseau par deux outputs pour classifier les images vraies ou fausses. Ils gardent donc le reste du réseau dont les paramètres sont pré-entraînés sur le dataset ImageNet [21].

Pour évaluer les performances de toutes ces techniques sur une tâche de détection de manipulation faciale, les auteurs de FaceForencics++ proposent une méthode de test automatique. Le but est de classifier comme vraies ou fausses des vidéos originales ou manipulées par les quatre techniques de manipulation précédemment citées. La difficulté de la tâche étant fortement liée à la qualité de l'image, **trois niveaux de compression** sont utilisés : sans compression, avec légère compression (paramètre de quantification fixé à 23) et forte compression (quantification à 40). Les résultats du benchmark sont visibles dans la figure 1. Le tableau contient le pourcentage de bonnes classifications, appelé précision, de chaque algorithme, sur les trois niveaux de compression. On voit bien que tous les algorithmes ont de très bons résultats sur les vidéos non-compressées. La véritable difficulté est donc bien dans la qualité de l'image, liée au niveau de compression de la vidéo.

À toutes les techniques étudiées dans FaceForencics++ [8], nous ajouterons le papier de Sabir et al. [22], sorti récemment, qui traite également de la détection de manipulations faciales. Dans ce papier, les auteurs proposent une nouvelle architecture utilisant des couches de convolution récurrentes permettant d'extraire de l'information temporelle. Elles sont donc particulièrement pertinentes dans le cadre d'une analyse de vidéos. Les auteurs présentent des résultats meilleurs que ceux de FaceForencics++, avec une amélioration de 4.55% en précision. Il nous a néanmoins été impossible de définir exactement sur quelles données, et donc quel niveau de compression, cette architecture a été testée.

Compression	Raw	HQ	LQ
Fridrich et al. [13]	97.63	70.97	55.98
Cozzolino et al. [15]	98.57	78.45	58.69
Bayar and Stamm [16]	98.74	82.97	66.84
Rahmouni et al. [17]	97.03	79.08	61.18
Afchar et al. [18]	95.23	83.10	70.47
Chollet [20]	99.26	95.73	81.00

Figure 1 : résultats du benchmark de FaceForencis++ sur les différents modèles de détection de manipulation faciale, sur trois niveaux de compression : sans (*"raw"*), faible (*High Quality = HQ)*, forte (*Low Quality = LQ*)

1.2. Développement

1.2.1. Détection de visage

Le développement de notre algorithme de détection de *deepfakes* s'est articulé autour de deux modèles : l'architecture Xception proposée dans le papier FaceForencics++ et un autre modèle dont nous avons conçu l'architecture. Mais avant de rentrer dans la recherche d'un *deepfake* en elle-même, une étape doit être réalisée : la détection des visages (voir figure 2). D'abord parce que notre projet se focalise sur les manipulations faciales, les vidéos ne contenant pas de visages ne seront donc pas traitées par DeepTruth. Ensuite parce que nos modèles de

détection de *deepfakes* ne regardent que la partie de la vidéo où se trouve le visage. Nous devons donc au préalable trouver la position des visages dans la vidéo et découper la partie de la vidéo correspondante. En plus de trouver la position des visages, une étape d'alignement est également réalisée. Pour cela, des points de repères communs à tous les visages (sourcils, bout du nez, yeux…) sont utilisés pour s'assurer que tous les visages extraits soient bien droits.

Cette étape de détection de visages a été développée deux fois, en parallèle : une fois en javascript, dans le cadre de l'extension web ; une seconde fois en python, dans le cadre de l'entraînement de nos modèles de détection de *deepfakes*. En javascript, nous utilisons la librairie face-api.js. Celle-ci utilise un CNN simple pour détecter la présence d'un visage dans une image. En python, nous utilisons la librairie dlib [23]. Elle implémente la méthode de Vahid Kazemi et Josephine Sullivan [24] qui utilise un arbre de régression en cascade pour détecter les points de repère correspondants à un visage. Une fois ces points de repère détectés, on peut les utiliser pour aligner le visage et découper la partie de l'image qui le contient.

Figure 2 : Illustration du processus de détection d'un *deepfake* [8], avec d'abord la détection et l'extraction d'un visage, puis la classification en "Vrai" ou "Faux"

1.2.2. Xception

Une fois les visages alignés et extraits, nous pouvons nous pencher sur leur classification. Nous cherchons à déterminer la probabilité qu'un visage soit vrai ou faux. Nous allons donc utiliser un modèle de *deep learning* pour estimer ces probabilités. Comme dit plus haut, nous avons utilisé deux de ces modèles au cours du développement. Le premier est celui présenté dans FaceForencics++. L'architecture **Xception** qu'il utilise est une excellente base pour classifier des images. De plus, les auteurs du papier ont publié le code du modèle ainsi que ses poids pré-entraînés. En théorie nous pouvons donc l'utiliser sans avoir à l'entraîner.

Ce modèle prend en entrée une image d'un visage au format RGB, d'une taille de 299 par 299 pixels. Le modèle Xception est basé sur l'architecture CNN. Il utilise donc des opérations de convolution pour extraire automatiquement des caractéristiques de l'image. Celles-ci sont ensuite utilisées par une couche dense de neurones artificiels à deux outputs pour prédire la classe de l'image. Les deux outputs correspondent donc respectivement à la probabilité que l'image soit vraie et la probabilité qu'elle soit fausse. Pour classifier une vidéo, 100 frames de celle-ci sont analysées par le modèle. On prend ensuite la moyenne de toutes les prédictions pour avoir la classe de la vidéo.

Pendant le développement de ce modèle, nous avons découvert le déroulement du ***Deepfake Detection Challenge*** (DDC) [25] : une compétition kaggle de détection de *deepfakes*. Celle-ci propose un autre dataset de vidéos manipulées plus volumineux et varié que celui de FaceForencics++. Cela nous a donc permis de tester le modèle Xception sur de nouvelles données. Le modèle s'est avéré très peu efficace : il classifiait mal les vidéos même en présence de *deepfakes* très visibles. Cela est très certainement dû aux données sur lesquelles le modèle est entraîné. Comme dit plus haut, le dataset de FaceForencics++ est composé de vidéos de journalistes de télévision : ils sont donc bien éclairés, parlent tous face à la caméra et ne bougent

quasiment pas. Le dataset du DDC est bien plus varié : les personnes bougent dans le cadre, ne regardent pas la caméra, tournent la tête et sont souvent mal éclairées. Ce constat remet donc en question le processus d'entraînement et le dataset de FaceForencics++. Néanmoins, nous pensons que l'architecture Xception a un bon potentiel pour notre tâche. Nous avons donc décidé de réentraîner le modèle sur les données du DDC. Après une première de phase de réentraînement sur une petite partie du nouveau dataset, le modèle a montré de belles performances, avec une précision atteignant 94%. Néanmoins, nous nous sommes rapidement rendu compte que cet entraînement était biaisé. En effet, les labels des données utilisées n'étaient pas du tout équilibré : 92% des vidéos étaient des *deepfakes*, ne laissant que 8% de vidéos originales. C'est un problème classique dans l'entraînement d'un modèle : si le dataset est déséquilibré, le modèle donnera des résultats en apparence bons pendant l'entraînement, mais très mauvais sur de nouvelles données de test. Nous avons réglé ce problème en prenant plus de photos dans les vidéos originales que dans celles manipulées. A l'heure de la rédaction de ce rapport, le modèle est en cours d'entraînement.

1.2.3. Notre modèle : 3DConvDetector

Devant les premiers résultats décevant du premier modèle face aux données du DDC, nous avons décidé de créer notre modèle, en parallèle du réentraînement du premier. Ce second modèle n'a pas la même approche. Notre intuition de base était similaire à celle du papier de Sabir et al. [22], précédemment cité. Pour détecter un *deepfake*, la vidéo contient des informations temporelles importantes telles que des micro-mouvements du visage ou des transformations de certaines parties du visage. Pour pouvoir utiliser ces informations temporelles il faut impérativement analyser plusieurs frames de la vidéo à la fois. Or, comme on l'a vu plus haut, le modèle Xception analyse les frames une par une en se basant uniquement sur les pixels de ces frames. Il n'a donc pas accès aux données temporelles visibles dans l'évolution des frames.

Pour pallier à ce problème, notre modèle, que nous appellerons **3DConvDetector**, utilise la convolution 3D. Cette technique réalise des opérations similaires à la convolution 2D utilisée dans Xception, mais en ajoutant la dimension du temps à l'analyse. Autrement dit, le modèle prend un très court extrait de la vidéo en entrée et en tire des informations spatiales et temporelles. L'inconvénient de cette approche est que la convolution 3D est beaucoup moins développée que sa version 2D. Nous sommes donc partis de quasiment rien pour créer notre modèle. Néanmoins, nous avons réussi à créer une architecture simple qui donne de bons résultats. Le modèle prend en entrée 10 frames consécutives de la vidéo au format RGB, d'une taille de 150 par 150 pixels. Il est composé de 8 couches de convolution 3D et d'une couche dense à une output. Cette output correspond à la probabilité que l'extrait analysé soit un *deepfake*.

Après l'avoir entraîné pendant 10 époques sur une petite partie du dataset (92 000 extraits de 10 frames), nous atteignons une précision de 95%. Un très bon résultat qui pourra être amélioré en continuant à s'entraîner sur le reste des données à notre disposition.

1.3. Résultats finaux

Les phases d'entraînement de nos deux modèles sont très longues : 19 heures pour 10 époques d'entraînement du 3DConvDetector sur les 92 000 extraits. Nous estimons que cette quantité d'extraits correspond à un vingtième de toutes nos données. Entraîner sur le reste va donc prendre beaucoup de temps. Nous nous contentons donc de ces résultats pour l'instant, mais nous continuons à entraîner les deux modèles dans l'espoir d'améliorer leurs

performances. Notre but est d'avoir un bon score à la compétition kaggle et d'avoir un modèle performant sur des vidéos variées. Pour l'instant, nous utiliserons notre 3DConvDetector mais, à terme, il faudra décider de quel modèle garder. La précision du modèle restera le facteur le plus important dans notre choix. Néanmoins, si nous avons deux modèles similaires en performances, il sera peut-être judicieux de prendre celui qui est le moins gourmand en puissance de calcul pour alléger nos besoins côté serveur et proposer une analyse plus rapide. Il sera également intéressant de suivre les résultats du DDC : si les gagnants publient leurs modèles, il sera judicieux de les utiliser dans DeepTruth.

2. Fingerprinting

Lorsqu'on parle de *fingerprinting*, on parle de fichiers multimédias dont on va extraire une empreinte qui correspond de façon unique au fichier d'origine [28]. Le *fingerprinting* pour les fichiers multimédias comme les images ou bien l'audio passe souvent par des fonctions de hachages. Ces fonctions vont réduire et synthétiser l'information dans le fichier vers un identifiant de taille fixe. Deux types de fonctions de hachage existent pour le *fingerprinting* : perceptuelle et non cryptographique. En effet, les fonctions de hachage cryptographique ne seraient pas adaptées au *fingerprinting* à cause de **l'effet avalanche** (aussi appelé principe de diffusion) qui fait qu'une petite modification du fichier d'origine entraîne une forte modification dans le *hash* de sortie (voir figure 3). En revanche, les fonctions de hachages perceptuelles n'opèrent pas selon ce principe car des fichiers relativement similaires auront des *hashs* également similaires. Cette particularité des fonctions de hachage perceptuelles font qu'elles sont robustes à de faibles modifications du fichier d'origine. Ceci est un point important pour les images car il est très facile de modifier le contenu d'origine de façon inaperçue afin de modifier le *fingerprint* du fichier. Cette robustesse aux modifications du contenu est importante pour notre cas d'usage car un *deepfake* est une modification du contenu d'origine. De plus, la modification dans les *deepfakes* est concentrée sur le visage des personnes, ce qui prend généralement peu de place sur la vidéo. On verra plus tard que cet avantage des fonctions de hachage perceptuelles peut aussi être une faiblesse, notamment pour les collisions de *hash*.

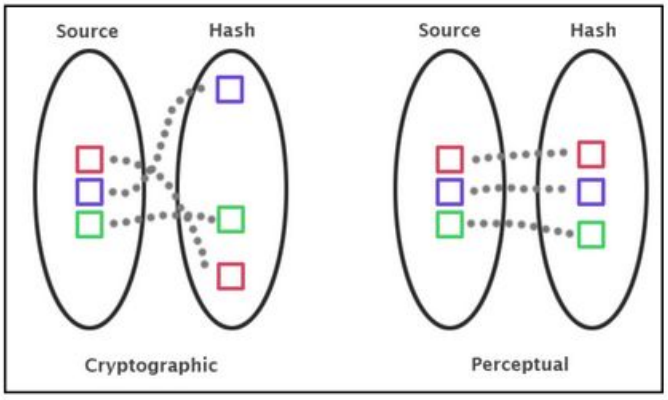

Figure 3 : Illustration des techniques de hachage cryptographique et perceptuelle

2.1 Etat de l'art

Depuis quelques années, on a vu beaucoup de développement dans le domaine des fonctions de hachage perceptuelles pour le contenu multimédia. Cette avancée est motivée par le besoin de pouvoir protéger les droits des entreprises sur le contenu qu'elles mettent en ligne. En effet, une fois mis en ligne, il est difficile de contrôler la distribution de contenu multimédia sur internet.

En terme d'algorithmes ou de fonctions de hachage, il existe un grand nombre d'implémentation différentes avec des performances variables. Certains algorithmes sont simples et rapides mais sont moins robustes aux modifications du contenu, tandis que d'autres implémentations sont coûteuses en calculs mais sont très robustes aux transformations. Les algorithmes simples se basent uniquement sur des comparaisons entre les valeurs des pixels tandis que ceux plus complexes utilisent des transformations mathématiques pour passer dans le domaine spectral.

2.2 Développement

Parmi toutes les librairies de *fingerprinting* disponibles nous avons retenu la librairie open source **pHash** [26]. Cette librairie a été codée en C++ pour une thèse sur une nouvelle technique de *fingerprinting*. Ce projet regroupe un grand nombre d'algorithmes de *fingerprinting* pour tout type de contenu (image, vidéo, audio). Nous avons fait le choix de cette librairie d'abord car c'est la plus souvent citée dans les projets utilisant du *fingerprinting*, mais aussi à cause de sa diversité d'algorithmes disponibles.

Il existe également une version pro de pHash qui propose de nouveaux algorithmes de *fingerprinting* ainsi que l'optimisation de certains autres, atteignant une vitesse de calcul jusqu'à 50 fois plus élevée. Néanmoins, la version classique est largement suffisante pour nos besoins.

2.2.1 Algorithme de *fingerprinting* : *DCT Hash*

L'algorithme de *fingerprinting* que nous avons choisi d'utiliser s'appelle *DCT Hash* [27][29]. Il utilise la transformée en cosinus discrète pour basculer l'image dans le domaine fréquentiel, d'où il en extrait l'empreinte. Cet algorithme est un bon compromis entre performance et une robustesse aux modifications du contenu. De plus, un des avantages des méthodes spectrales est qu'elles sont très tolérantes aux algorithmes de compression (mpeg, h264). Voici une courte explication du fonctionnement de l'algorithme, illustrée par la figure 4 :

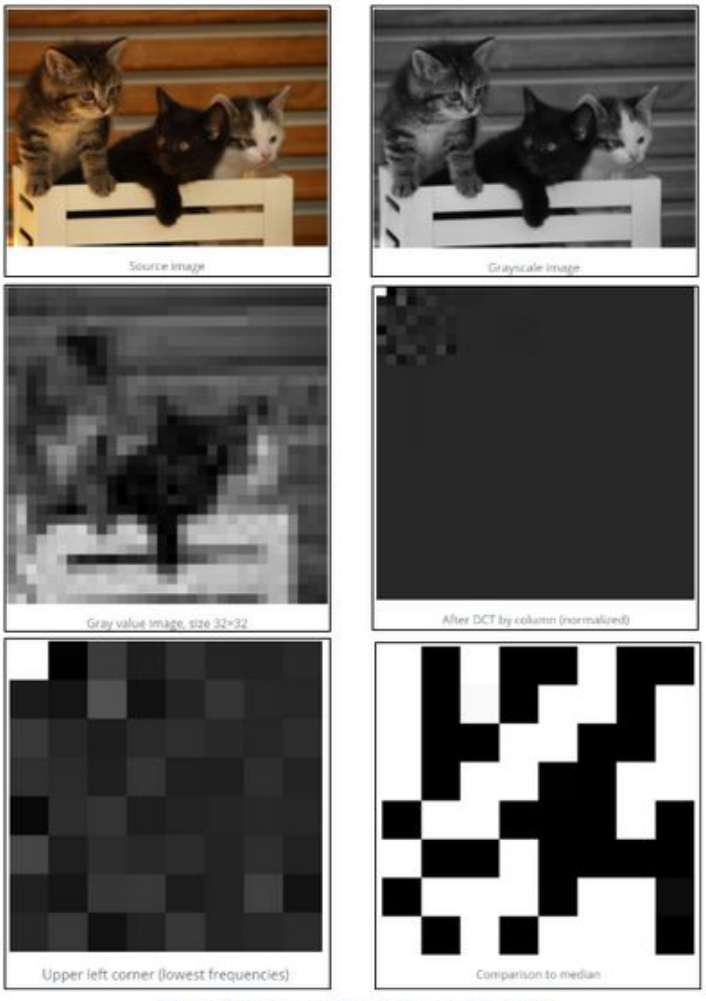

Figure 4 : Etapes de l'algorithme *DCT Hash*

1) Passage en noir et blanc : afin de diminuer les dimensions de l'image, on la passe en noir et blanc. Ceci a aussi l'effet d'ignorer les modifications de couleur qu'on aurait pu faire, comme la correction gamma.

2) Redimensionnement de l'image : on réduit encore les dimensions de l'image à une taille de 32 par 32 pixels. La taille choisie est arbitraire.

3) Transformée cosinus discrète (DCT) : On applique la transformée en

cosinus discrète sur l'image. On remarquera qu'après la normalisation, il ne reste qu'un carré de 8x8 en haut à gauche qui correspond aux composantes basses fréquences extraites par la DCT. On utilisera ces basses fréquences pour construire notre *hash*.

4) Extraction des basses fréquences (coin en haut à gauche) : Comme expliqué plus haut, on garde uniquement les basses fréquences de l'image d'origine. Les composantes de plus hautes fréquences ne seraient pas utiles dans la construction du *hash* car elles sont plus susceptibles de changer lorsqu'on modifie l'image.

5) Construction du *hash* : comparaison à la médiane : Enfin, pour la dernière étape il faut pouvoir extraire une chaîne binaire de taille 64. Pour ce faire, on calcule la médiane des composantes basses fréquences extraites. Ensuite pour chaque "pixel" restant, on mettra sa valeur finale à 1 ou à 0 selon s'il est supérieur ou inférieur à la valeur médiane calculée. Il ne reste plus qu'à définir dans quel ordre on arrange les "pixels" pour construire notre *hash*.

Cette méthode pour calculer le *hash* est très similaire aux autres algorithmes [30]. Le choix des dimensions de l'image peut changer selon la taille du hash qu'on souhaite avoir en sortie. Nous utilisons une hash de 64 bits, il nous faut donc une image de 8 par 8 "pixels" à l'étape 4.

2.2.2 Calcul de distance entre les *hashs* : Distance de Hamming

Maintenant que savons calculer le *hash* d'une image quelconque, il faut trouver une manière de comparer les *hashs* entre eux. Il existe plusieurs métriques pour ce faire, nous utiliserons la plus simple: la distance de Hamming. Cette distance donne le nombre de permutations à effectuer sur une chaîne binaire afin de passer d'une chaîne à l'autre. Comme on peut le voir sur la figure 5, chaque arête du cube correspond à une unité de la distance de Hamming. Plus les deux chaînes sont différentes, plus leur distance sera grande. Dans notre cas, une distance inférieure à 12 nous indique qu'il y a une forte chance que les deux *hashs* correspondent à la même image.

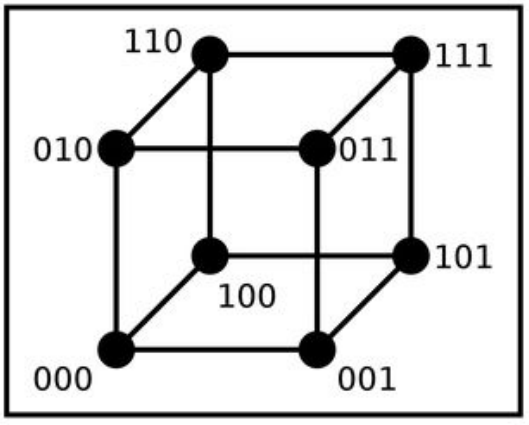

Figure 5 : Visualisation de la distance de Hamming

2.2.3 Stockage des *hashs*

Après avoir déterminé comment calculer et comparer les *hashs*, il faut établir quel genre de base de données sera utilisée pour les stocker. Les *hashs* que nous calculons avec la librairie pHash font 64 bits de long et nous avons uniquement besoin de savoir si la vidéo est un *deepfake* ou non. Dans ce cas, nous aurons besoin de deux colonnes : une pour le *hash* et une seconde pour le label de la vidéo ("Vraie" ou "*Deepfake*"). Cependant, il reste un problème lié aux *hashs* : les collisions. En effet, il est possible que des images se ressemblent fortement et que la distance de Hamming soit inférieure au seuil établi. De plus, puisqu'on essaie de détecter des *deepfakes* il faut pouvoir faire la différence entre les *hashs* de la vidéo originale et de la version *deepfake*. Or, lors des tests que nous avons réalisé, nous avons remarqué qu'il est impossible de faire la différence entre la vidéo originale et la version *deepfake*. Ceci est dû au fait que la version *deepfake* ne change que faiblement le contenu global de la vidéo. C'est seulement le visage qui est modifié. Il est donc normal d'avoir ce genre de collisions. Ceci pose un problème pour le

fonctionnement de notre extension car il ne faut pas juste savoir si on a vu cette vidéo auparavant, mais si cette vidéo est un *deepfake* ou non.

Puisqu'on ne peut pas juste utiliser le hash de la vidéo pour identifier la vidéo et son label, il faut ajouter une vérification supplémentaire pour faire la différence entre les vidéos. Notre solution est de calculer le *hash* de la partie de l'image qui contient le visage. Cette partie étant différente entre une image originale et son *deepfake*, leur *hash* seront différents. On peut donc les utiliser pour identifier les vidéos dans la base de données.

La recherche se fera donc en deux temps : dans un premier temps on identifie si la vidéo en question existe dans notre base de données. Dans un deuxième temps, on effectue une recherche sur le visage extrait de la vidéo. Avec ces deux recherches successives on peut identifier la vidéo et récupérer son label.

Pour le type de base de données à utiliser, le plus pratique sera une base NoSQL afin de pouvoir stocker de manière imbriquée les *hashs* des vidéos ainsi que les *hashs* des visages contenus dans la vidéo avec leur label. Parmi les différentes bases NoSQL disponibles nous avons choisi **MongoDB**. MongoDB est une base orientée document que nous avions utilisé avant ce qui a rendu la mise en place et l'usage plus rapide. C'est aussi une base de données qui peut fonctionner en mode distribué. Elle est donc facilement scalable ce qui sera utile pour suivre l'évolution des besoins de notre projet. De plus, MongoDB permet de faire de l'indexation sur un champ en particulier du document. En faisant l'indexation sur le champ du *hash* de la vidéo, on peut fortement optimiser la recherche sur ce champ en particulier.

3. Extension web

Une extension de navigateur (aussi appelée "extension web") permet d'ajouter des fonctionnalités supplémentaires au navigateur de l'utilisateur. Les extensions de navigateur existent afin de personnaliser la navigation sur le web pour la rendre plus agréable ou bien l'adapter aux besoins de chacun. La volonté de notre projet est de permettre aux utilisateurs qui regardent des vidéos sur Internet de pouvoir prendre du recul sur la véracité du contenu qu'ils sont en train de visionner. Comme nous l'avons vu précédemment, il y a un essor des algorithmes permettant la modification de l'apparence ou du comportement d'une personne présente dans une vidéo. Notre outil DeepTruth permet à chaque internaute de s'informer sur le contenu qu'il regarde et de pouvoir démêler le vrai du faux. Malgré l'émergence des *deepfakes* sur Internet, il n'existe actuellement aucune extension de navigateur permettant de détecter leur présence. Notre projet cherche donc à répondre à ce manque.

3.1. Développement

3.1.1. Fonctionnement de l'extension

Pour les extensions web qui nécessitent des actions de la part de l'utilisateur, on dispose de vues, qui sont des pages web miniatures couplées à des scripts afin de gérer les différentes interactions. En revanche, toutes les extensions web ne nécessitent pas l'action d'un l'utilisateur pour pouvoir fonctionner, comme c'est le cas pour DeepTruth. L'extension utilise alors des scripts qui s'exécutent sans que l'utilisateur ne les voient, mais qui peuvent tout de même modifier le contenu de la page. Une extension web se construit autour d'un fichier clé nommé "manifest.json". Il définit l'architecture globale de l'extension : le nom, la description, les icônes

qui seront affichées et les différentes vues et scripts qui seront utilisés. Les vues correspondent à des documents HTML affichés lorsque l'utilisateur clique sur l'icône de l'extension dans son navigateur. Les scripts quant à eux permettent d'exécuter une grande variété de tâches différentes. Il existe trois principaux types de scripts : les scripts liés aux vues, les scripts de contenu et les scripts d'arrière-plan. La complexité dans le développement d'une extension web réside dans la gestion de ces trois composants. Les scripts liés aux vues seront exécutés uniquement lorsqu'un utilisateur interagit avec une vue. Les scripts de contenu eux, seront exécutés lorsqu'une page web est chargée par le navigateur. Enfin, les scripts d'arrière-plan définissent des *"event listeners"* qui lanceront l'exécution de code lors de l'occurrence d'un événement spécifié.

Figure 6 : Fonctionnement de l'extension DeepTruth

Penchons-nous maintenant sur le fonctionnement interne de l'extension DeepTruth. La figure 6 en présente un schéma simplifié que nous allons expliquer étape par étape :

1) Détection d'une vidéo en cours de lecture : DeepTruth attend l'ouverture d'une nouvelle page web sur le navigateur pour pouvoir se lancer. Dès lors qu'une page web est ouverte, un script de contenu va s'exécuter afin d'analyser toutes les vidéos présentes dans la page. Le script utilise la balise HTML *<video>* et évalue la valeur de ses attributs *paused* et *currentTime* pour déterminer son état. Ce script de contenu permet de détecter si l'une d'elle est en train d'être lue par l'utilisateur.

2) Extraction des visages et envoi au serveur : Si le script de contenu détecte une vidéo en cours de lecture, alors il lance le processus d'extraction des visages. Ce processus, réalisé par un script javascript, permet de récupérer le visage de chaque personne présente dans la vidéo lue. Chaque visage détecté est ensuite enregistré sous forme d'image, puis celle-ci est envoyée au serveur. Cette étape est la plus importante dans le cycle de traitement de l'extension web, et ce, pour deux raisons principales. La première concerne les performances : c'est l'étape la plus gourmande en ressources. Or tous les traitements effectués par une extension web sont gérés par l'ordinateur de l'utilisateur. Cela peut causer des ralentissements si les traitements de l'extension web ne sont pas optimisés. En plus d'améliorer les performances, il faut garder une précision maximale pour que l'analyse fonctionne.

3) Réponse du serveur et marquage de la vidéo : Après avoir envoyé les visages extraits, le serveur envoie une réponse indiquant si la vidéo est un *deepfake* ou non. L'extension analyse le contenu de la réponse du serveur et réagit en conséquence. S'il s'agit d'un *deepfake*, alors l'extension remplace la vidéo par un message d'alerte indiquant à l'utilisateur qu'il s'agit d'une vidéo falsifiée. Si la réponse est négative, l'extension n'effectue aucune action. Lorsque l'utilisateur voit le message d'alerte, il peut le masquer et reprendre la lecture de sa vidéo en cliquant sur un bouton "Voir le contenu".

3.1.2. Techniques utilisées

L'extension web est programmée en html, css et javascript. Il est primordial de sélectionner un navigateur sur lequel sera déployé le premier prototype de l'extension. En effet, d'un navigateur à l'autre l'architecture du projet peut varier. Dans notre cas, nous avons choisi le navigateur Chrome de Google car c'est le navigateur avec le plus utilisé avec plus de 60% de part de marché.

3.1.3. Difficultés rencontrées et solutions

Dès la phase de conception de l'extension web nous avons rencontré plusieurs difficultés. La première résidait dans l'élaboration d'une méthode de détection d'une vidéo en cours de lecture fonctionnant sur le plus grand nombre de sites web possible. Cette difficulté s'explique par la liberté de conception apportée par le langage HTML5 (principal langage utilisé dans le développement de sites web). En effet, HTML5 est un langage de balisage qui permet au développeur de créer une page web en suivant l'architecture qui lui semble être la plus pertinente. Cela amène une grande diversité dans le contenu des différentes pages, ce qui complexifie la création d'une méthode de détection "universelle". Pour résoudre ce problème, nous avons cherché s'il existe un élément HTML commun à toutes les pages web qui contiennent une vidéo. Cet élément existe et c'est la balise <video> du langage HTML5 qui permet d'intégrer un contenu vidéo. Une fois cet élément identifié, nous avons pu mettre en place une boucle pour rechercher toutes les vidéos présentes sur la page et ainsi détecter si l'une d'elle est en train d'être lue par l'utilisateur.

Au niveau du module javascript de détection de visage (face-api.js), la première difficulté a été de modifier légèrement les fonctionnalités proposées par celui-ci. En effet, cette API est plutôt destinée à l'analyse d'images. Cependant, il est indiqué dans la documentation qu'il est possible d'utiliser certaines fonctions telles que le face tracking - détection de visage en temps réel lors d'une vidéo. Nous avons donc pu modifier cette fonction afin d'extraire chaque visage détecté.

Une autre difficulté qui nous est apparue est que cette API est faite pour fonctionner sur un serveur node.js, mais une extension web n'utilise que du javascript classique. Node.js étant basé sur javascript, il a été possible de modifier légèrement le code du module afin de l'implémenter dans l'extension simplement. Cependant, une des fonctions doit accéder à des fichiers binaires contenant les poids du modèle de détection de visage, et nous ne pouvons pas inclure ces fichiers dans l'extension. Pour contourner ce problème, nous faisons en sorte que l'extension envoie une requête à un serveur en local.

La dernière difficulté rencontrée est un problème de "*crossorigin*". Notre extension utilise les vidéos lues dans le navigateur pour en extraire les images. Cependant, les navigateurs ne nous laissent pas faire cela si l'origine de la vidéo n'est pas reconnue et qu'elle ne provient pas du même domaine que le site web. Cette fonctionnalité existe pour renforcer la sécurité informatique en évitant qu'un site charge des documents potentiellement malicieux depuis un autre site. Ce problème étant arrivé pendant les tests finaux, nous n'avons pour le moment trouvé aucune solution pour le régler. Notre extension fonctionne donc très bien sur des vidéos stockées sur le site web visité mais pas sur des celles d'origines externes.

4. Produit fini

4.1. *Proof of Concept*

Les trois axes de notre projet se rejoignent dans notre *Proof of Concept* (POC). Son fonctionnement est résumé dans le schéma d'architecture de la figure 7. L'extension web se charge de détecter la lecture d'une vidéo. Si un visage est détecté dans celle-ci, il est extrait et envoyé à notre serveur, en local pour notre POC. Quand le serveur reçoit les frames d'un visage, il les enregistre dans un dossier. Le script de *fingerprinting* est alors lancé pour calculer le hash de ce visage et le rechercher dans la base de données. S'il a été déjà été analysé par DeepTruth, il sera enregistré dans la base avec son label : "vrai" ou "*deepfake*". Le résultat peut donc être directement renvoyé à l'utilisateur. Par contre, si on ne retrouve pas la vidéo dans la base, le script de détection de *deepfake* est lancé. Il prend les frames extraites, les traite pour qu'elles puissent être lues par notre modèle et les envoie à celui-ci. La prédiction de notre modèle est stockée dans la base de données pour rendre une prochaine analyse de cette vidéo plus rapide. La prédiction est également renvoyée à l'extension qui affiche une alerte en cas de *deepfake*.

Dans l'état, nous sommes capable de réaliser une analyse d'une nouvelle vidéo en moins de 15 secondes. Si la vidéo a déjà été visionnée par DeepTruth, et donc contenue dans notre base, le temps d'analyse descend en-dessous de 2 secondes. Le *fingerprinting* nous apporte donc bien un gain considérable de temps et de puissance de calcul.

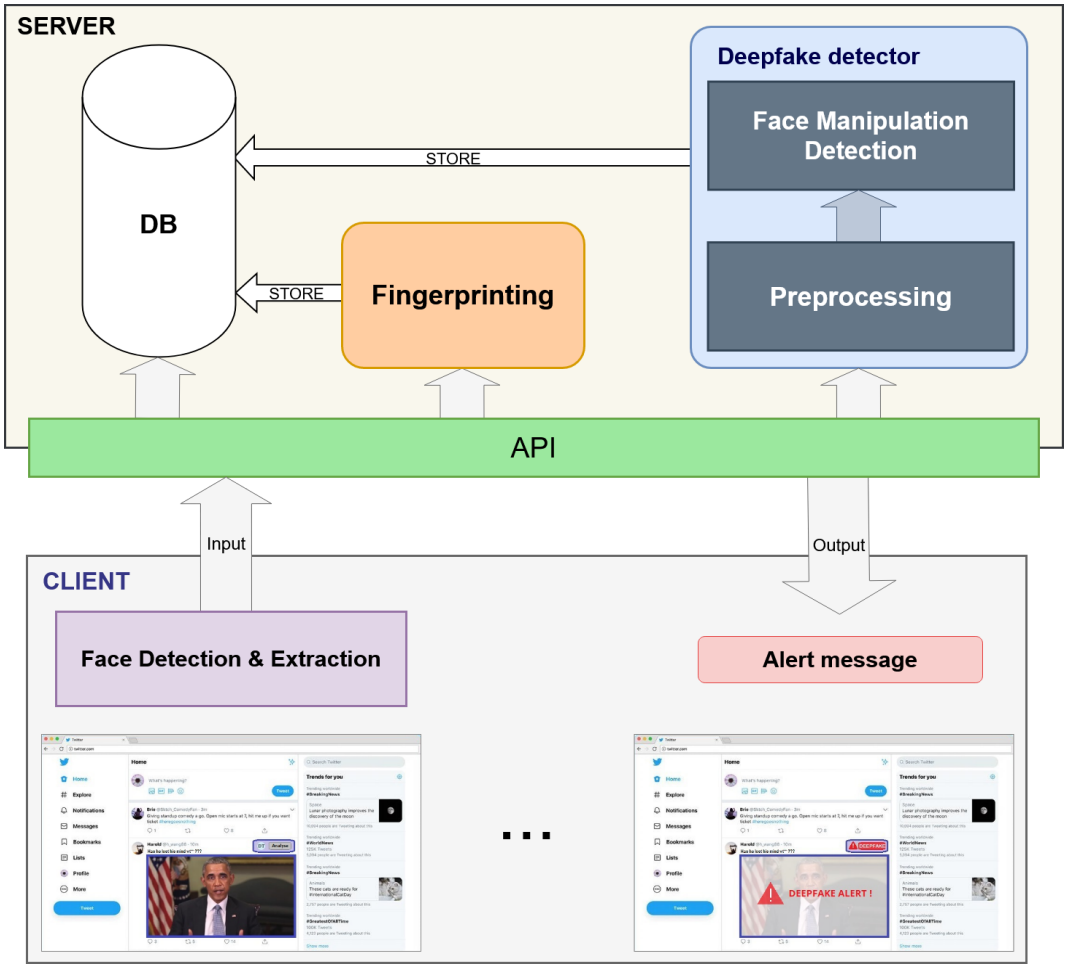

Figure 7 : Architecture globale de DeepTruth

4.2. Suites du développement

Dans le but de mettre en production ce POC, plusieurs étapes devront être réalisées. D'abord, nous devrons mettre en place un vrai serveur, accessible par tous. Notre partenaire Nextino nous a proposé d'avoir accès à des machines et de l'espace de stockage dans leurs locaux. Le serveur et la base de données associée pourront donc y être installés. Ce serveur devra être assez puissant pour faire tourner notre modèle de détection de *deepfakes*. L'idéal serait d'avoir accès à un processeur graphique (GPU), particulièrement adéquat pour faire tourner des modèles de *deep learning*. Néanmoins, de bons processeurs classiques feront également l'affaire dans le cas d'une inférence sur un modèle déjà entraîné. En terme de mémoire, le serveur devra avoir au moins 8 GO (1 GigaOctet $\simeq 10^9$ octets) de mémoire vive pour soutenir les calculs de notre modèle. Les informations que nous stockons sont pour l'instant très minces : le *hash* d'un visage et la probabilité qu'il soit un *deepfake* tiennent sur 12 octets. Une base de seulement 10 GO nous permettra de stocker près d'un milliard d'entrées. Néanmoins, il est possible que nous décidions de stocker d'autres données en plus dans le futur. Nous pourrions, par exemple, garder en mémoire le nom de la vidéo, son lien, ou d'autres informations qui rendraient notre analyse plus rapide ou plus précise. Dans ce cas une entrée dans la base serait un peu plus volumineuse. Mais rien de particulièrement lourd ne serait stocké, une petite base de données devrait donc nous convenir.

L'extension web en elle-même nécessite plusieurs développements. Naturellement notre première étape sera de publier l'extension sur le Chrome Web Store pour qu'elle soit accessible aux nombreux utilisateurs de Chrome. Dans un second temps, il faudra adapter l'extension aux autres navigateurs très utilisés comme Safari, Firefox, Edge ou encore Opera. Nous prévoyons également d'ajouter des fonctionnalités de paramétrage de l'extension, avec des options comme l'analyse automatique de toutes les vidéos lues ou le blocage de l'extension sur certains sites.

La partie *fingerprinting* de vidéos pourra également être améliorée. En effet, il est important de noter que les durées d'analyse évoquées plus haut sont susceptibles de changer lors de la mise en production de notre projet. Plus on analyse de vidéos, plus la base de données a d'entrées. L'étape de recherche des *hashs* dans la base prendra donc de plus en plus de temps. Cela pourra être compensé par une étape d'optimisation de notre algorithme de comparaison des *hashs*.

Enfin, comme nous l'avons dit dans la première partie de ce rapport, nos modèles de détection de *deepfakes* doivent encore être entraînés. Ils peuvent également être améliorés en fonction de nos observations ou des ressources publiées sur le *Deepfake Detection Challenge* [25].

5. Partenariat : Nextino

Pour la valorisation de ce projet, nous avons effectué un partenariat avec le centre de recherche et développement en IA Nextino. Cette startup fait partie du groupe Atempo-Wooxo, s'intégrant comme leur structure d'innovation et de recherche. Atempo-Wooxo fournissent des produits de sécurisation et d'archivage de données et sont aujourd'hui des acteurs reconnus dans la lutte contre la cybercriminalité.

Depuis quelques années, la RGPD (Réglementation Générale sur la Protection des Données) est entrée en vigueur et les entreprises européennes doivent se responsabiliser sur le domaine de la protection des données sensibles. Atempo-Wooxo ayant un accès privilégié aux

données de leurs clients, il est important pour eux de développer des outils RGPD afin de protéger les intérêts de leurs clients. C'est pour cela que depuis 12 mois, Nextino a entrepris un projet de classification de contenu RGPD. Leurs premiers efforts se sont focalisés sur l'analyse de documents avec les technologies d'OCR (Optical Character Recognition) et de NLP (Natural Language Processing). Il leur manquait une autre partie qui est importante avec la RGPD qui est le droit à l'image.

Au départ, nous avions contacté Nextino pour leur proposer un projet alliant l'IA avec la cybersécurité. Ce projet a vite évolué vers la détection de *deepfakes* qui a été validé par notre partenaire. L'intérêt de Nextino dans ce projet n'était pas l'extension web en elle-même mais les différentes briques que nous avions développées. Elles s'intégreraient à leur projet de détection de contenu RGPD. En effet, les briques de détection et d'extraction de visages ainsi que la partie sur le *fingerprinting* sont très intéressantes pour enrichir la partie de détection de droit à l'image du projet de Nextino. C'est pour cela que nous avons particulièrement pousser la partie sur le *fingerprinting* afin d'explorer les possibilités à la fois dans le contexte de notre projet mais aussi celui de Nextino. Un rapport détaillé sur le *fingerprinting* a été fait avec l'explication des algorithmes et les cas d'usages possibles du *fingerprinting*. Il sera restitué à Nextino avec les modules de code qui les intéressent.

De leur côté Nextino ont pu nous fournir l'infrastructure nécessaire pour développer nos modèles de *deep learning* et héberger notre application. Ils ont un partenariat avec DiRAC (Distributed Research using Advanced Computing) qui est un centre de calcul haute performance où il est possible d'entraîner nos modèles de *deep learning* qui peuvent demander beaucoup de puissance de calcul et de mémoire. De plus, nous avons pu bénéficier de l'expertise de notre partenaire qui nous a conseillé sur les technologies existantes ainsi que les cas d'usage possibles.

Conclusion

Tout au long du développement de DeepTruth nous avons fait en sorte de répondre au mieux aux besoins initiaux en fiabilité, rapidité et universalité. D'abord en cherchant le meilleur modèle possible pour détecter un *deepfake*. Travailler sur plusieurs architectures nous a permis de mettre toutes les chances de notre côté pour avoir un modèle fiable. De plus, le développement et l'entraînement de ces modèles ne s'arrête pas maintenant, puisque nous utiliserons les retours du *Deepfake Detection Challenge* pour les améliorer. Ce travail sur des modèles complexes de *Computer Vision* nous a également apporté une expérience très enrichissante dans ce domaine de l'IA, qui nous sera certainement utile dans nos carrières futures.

Les techniques de *fingerprinting* de vidéos étaient pour nous complètement nouvelles. Nous avons réussi à nous les approprier et à en tirer profit pour rendre notre analyse beaucoup plus rapide. Le rapport sur le *fingerprinting*, que nous rendrons à Nextino, nous a permis de plonger au fond de ce domaine pour en comprendre au mieux tous les aspects.

Enfin, le travail sur l'extension web a posé plus de problèmes que prévu. Néanmoins nous avons une base solide qui peut simplement être étoffée. Ce principe d'universalité de notre solution nous est très cher. Nous sommes parfaitement conscients de la supériorité technique des grands groupes que nous citions dans l'introduction. Mais l'approche d'une extension web nous permet d'amener une solution pragmatique qui fonctionne partout sur internet.

Le travail avec notre partenaire Nextino s'est très bien déroulé. Nous remercions M. Bernard Peultier pour son suivi du projet et ses nombreux conseils. Nous espérons que notre travail trouvera son utilité dans les nombreux projets de Nextino. Dans cette optique, nos développements ont été réalisés de la manière la plus modulaire possible, et une documentation détaillée sera fournie à notre partenaire.

Ce PFE nous a donc apporté à tous une expérience technique dans le milieu de l'authentification de vidéos. Mais il nous a également permis d'enrichir notre vision du combat contre les *deepfakes*, et plus largement contre la désinformation. Il est clair que les *deepfakes* ne cesseront de s'améliorer. Dans cette optique, nous ne pouvons pas être sûrs qu'il sera toujours possible d'authentifier les vidéos que nous visionnons sur le web. Au-delà du combat technique, il y a donc une problématique sociale majeure résidant dans notre capacité à remettre en question constamment le contenu de la toile. Or, pour répondre à ce problème, il ne suffit pas de déceler les mensonges, mais bien d'éduquer les utilisateurs du web à l'esprit critique et à la remise en question. Si nous ne sommes pas capables de détecter les meilleurs *deepfakes* que l'avenir nous réserve, nous espérons donc que DeepTruth et les projets similaires permettront d'amener une prise de conscience globale des utilisateurs sur la présence de fausses informations sur le web.

Annexes

1. Techniques de manipulation faciale

FaceSwap [10] est une approche graphique qui se base sur l'extraction de visage grâce à la détection de points de repère : coins des yeux, bout du nez, etc. Ces points servent à l'application d'un modèle 3D projeté sur l'image cible en minimisant les différences entre le masque et l'image aux abords des points de repère.

DeepFakes [11] a donné son nom aux techniques de manipulation faciales, mais il désigne en réalité une de ces méthodes. Elle repose sur deux auto-encodeurs partageant le même encodeur. Le but de l'auto-encodeur est de reconstruire en sortie le visage donné en entrée. Afin de réaliser un faux il suffit d'utiliser l'encodeur et le décodeur entrainés sur l'image source en lui passant en entrée le visage cible. Le résultat est ensuite traité à l'aide de bibliothèques d'édition d'image.

Face2Face [9] est un système de reconstitution faciale. Il se focalise sur le transfert des expressions d'un visage à l'autre, sans modifier son apparence. La méthode repose sur la sélection d'images précises dans les vidéos des deux visages. Celles-ci sont utilisées pour produire une reconstruction de l'image. Ces modèles peuvent être re-synthétisés afin de recréer le visage source sous différentes illuminations et avec de nouvelles expressions selon certains paramètres. Afin de générer un faux il transférer les paramètres relatifs aux expressions du visage source vers la vidéo cible, et ce sur chaque image de la vidéo.

NeuralTextures [12] se base sur l'utilisation d'un *Generative Adversarial Network* (GAN) afin de concevoir un système de reconnaissance faciale. Le GAN est une architecture très performante pour générer du nouveau contenu. Ici, il utilise la vidéo originale pour apprendre les "textures" du visage cible et s'entraîne à les reconstruire avec précision. Afin de créer un faux les auteurs utilisent un système de repérage pour suivre les mouvements du visage cible et ainsi créer les textures correspondantes. Il est ensuite possible de modifier ces textures pour animer le visage autrement en se calquant sur les mouvements d'un acteur, par exemple.

Références

[1] https://newsyoucantuse.com/
[2] https://www.thefakenewsgenerator.com/
[3] "*You Won't Believe What Obama Says In This Video!*"
https://www.youtube.com/watch?v=cQ54GDm1eL0
[4] "*Deepfakes* porn has serious consequences", Dave Lee, 3 février 2018 :
https://www.bbc.com/news/technology-42912529
[5] "*Twitter drafts a deepfake policy that would label and warn, but not always remove, manipulated media*", Sarah Perez, 11 novembre 2019 :
https://techcrunch.com/2019/11/11/twitter-drafts-a-deepfake-policy-that-would-label-and-warn-but-not-remove-manipulated-media/
[6] "*Creating a data set and a challenge for deepfakes*", Mike Schroepfer, 5 septembre 2019 :
https://ai.facebook.com/blog/deepfake-detection-challenge/
[7] https://trusted-news.com/
[8] *"FaceForensics++: Learning to Detect Manipulated Facial Images"* by Andreas Rössler et al., 2019 :
https://arxiv.org/abs/1901.08971
[9] *"Real-time expression transfer for facial reenactment"* par Justus Thies et al., 2015 :
https://dl.acm.org/citation.cfm?id=2818056
[10] *FaceSwap* github : https://github.com/MarekKowalski/FaceSwap/
[11] *DeepFakes* github : https://github.com/deepfakes/faceswap
[12] *"Deferred Neural Rendering: Image Synthesis using Neural Textures"* par Justus Thies et al., 2019 :
https://arxiv.org/abs/1904.12356
[13] *"Rich Models for Steganalysis of Digital Images"* par Fridrich et al., 2012 :
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.441.6997&rep=rep1&type=pdf
[14] *"Image forgery detection through residual-based local descriptors and block-matching"* par Davide Cozzolino et al., 2014 :
https://projet.liris.cnrs.fr/imagine/pub/proceedings/ICIP-2014/Papers/1569902439.pdf
[15] *"Recasting residual-based local descriptors as convolutional neural networks: an application to image forgery detection"* par Cozzolino et al., 2017 : https://arxiv.org/abs/1703.04615
[16] *"A deep learning approach to universal image manipulation detection using a new convolutional layer"* par Bayar et Stamm, 2016 :
http://misl.ece.drexel.edu/wp-content/uploads/2017/07/Bayar_IHMMSec_2016.pdf
[17] *"Distinguishing computer graphics from natural images using convolution neural network"* par Rahmouni et al., 2017 :
http://www-igm.univ-mlv.fr/~vnozick/publications/Rahmouni_WIFS_2017/Rahmouni_WIFS_2017.pdf
[18] *"Mesonet: a compact facial video forgery detection network"* par Afchar et al., 2018 :
https://arxiv.org/abs/1809.00888
[19] *"Inception-v4, inception-resnet and the impact of residual connections on learning"* par Szegedi et al., 2016 : https://arxiv.org/abs/1602.07261
[20] *"Xception: Deep Learning with Depthwise Separable Convolution"* par François Chollet, 2016 :
https://arxiv.org/abs/1610.02357
[21] ImageNet dataset : http://www.image-net.org/
[22] *"Recurrent Convolutional Strategies for Face Manipulation Detection in Videos"* par Sabir et al., 2019 :
https://arxiv.org/abs/1905.00582
[23] Dlib : http://dlib.net/
[24] *"One Millisecond Face Alignment with an Ensemble of Regression Trees"* par Vahid Kazemi et Josephine Sullivan, 2014 :
https://www.researchgate.net/publication/264419855_One_Millisecond_Face_Alignment_with_an_Ensemble_of_Regression_Trees
[25] *Deepfake Detection Challenge* : https://www.kaggle.com/c/deepfake-detection-challenge/overview
[26] pHash, librairie open source de fingerprinting pour fichiers multimédias: https://www.phash.org
[27] *"Perceptual Video Hashing for Content Identification and Authentication"* par Khelifi et al., 2019 :
https://ieeexplore.ieee.org/document/8116689
[28] digital guarding, what is file fingerprinting:
https://digitalguardian.com/blog/what-file-fingerprinting

[29] hackerfactor, pHash explanation:
http://www.hackerfactor.com/blog/index.php?/archives/529-Kind-of-Like-That.html
https://www.hackerfactor.com/blog/?/archives/432-Looks-Like-It.html
[30] okcupid blog post: evaluating different hash functions:
https://tech.okcupid.com/evaluating-perceptual-image-hashes-okcupid/