INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

SAFE GREYC

Rapport de stage 2ème année de cycle ingénieur

Implémentation d'un algorithme de classification binaire pour la détection de falsification vidéo

Effectué au laboratoire GREYC

du 11 octobre 2021 au 21 janvier 2022

Encadrants : M. Christophe CHARRIER et M. Emmanuel GIGUET
Tuteur enseignant : M. Youcef IMINE

Rédigé par

Paul CANCHON

Table des matières

Remerciements

Je témoigne ma profonde reconnaissance à M. Christophe CHARRIER pour avoir accepté de m'accueillir dans son équipe pendant mon stage, pour ses explications ainsi que le pour le temps qu'il m'a consacré.

Mes remerciements vont ensuite à M. Emmanuel GIGUET pour avoir coencadré mon stage mais aussi pour ses précieux conseils et sa disponibilité.

Je tiens aussi à remercier M. Hugo MERLY pour l'aide apportée durant mon stage.

Enfin, je remercie l'ensemble de l'équipe SAFE pour leur bon accueil au sein du laboratoire.

Résumé

Ce rapport présente les travaux réalisés lors de mon stage dans le laboratoire GREYC, à Caen. Le sujet porte sur la détection de falsifications inter-images dans une vidéo utilisant le format de compression H264. L'objectif est la mise en place d'un algorithme d'apprentissage automatique pour classer les vidéos en deux catégories falsifiées/non-falsifiées, à partir d'informations récupérées depuis le bitstream.

Nous détaillons d'abord la structure d'une vidéo ainsi qu'une partie de la compression H264, puis nous dressons un état de l'art des méthodes de détection de falsification vidéo. Nous présentons ensuite les moyens et les méthodes utilisées pour mettre en place l'algorithme d'apprentissage et la détection des vidéos falsifiées. Cette partie est divisée en trois étapes à savoir, la création d'un corpus de vidéo falsifiées/non-falsifiées, l'extraction des caractéristiques depuis le bitstream de la vidéo et enfin l'implémentation de l'algorithme. Finalement, nous présentons les résultats des tests réalisés.

Le corpus contient 555 vidéos dont 111 non-falsifiées et 444 falsifiées. Les falsifications sont réparties en 111 insertions, 111 suppressions, 111 duplications et 111 swaps. Les résultats semblent encourageants puisque notre méthode permet d'obtenir un taux d'exactitude de 90% lors de la classification de l'ensemble des vidéos. Ce taux diminue cependant quand la classification est réalisée sur un type de falsification particulière.

Chapitre 1

Introduction

1.1 Préambule

Aujourd'hui, avec l'avènement d'Internet et des réseaux sociaux, l'accès et la diffusion de l'information sont devenus de plus en plus simples et de plus en plus rapides, à tel point qu'on parle aujourd'hui de sur-information. Et bien qu'il soit plus facile d'avoir confiance en des preuves visuelles comme des vidéos, les outils d'édition se sont eux aussi démocratisés permettant à n'importe qui de les utiliser. Un exemple notable est la vidéo réalisée en 2018 par Jordan Peele, cinéaste américain, dans laquelle on voit l'ancien président des Etats-Unis, Barack Obama, injuriant Donald Trump. Cette vidéo est en réalité fausse et a été réalisée à partir d'un deepfake, une technique de falsification consistant à utiliser des techniques d'intelligence artificielle pour réaliser des falsifications de vidéos. Une autre utilité des vidéos aujourd'hui est de servir de preuve devant la justice. Cependant, il est là aussi impératif de pouvoir vérifier leurs authenticités et leurs intégrités afin qu'elles puissent être utilisées.

Dans ce contexte, il est important de disposer d'outils efficaces afin de pouvoir déterminer si une vidéo a été falsifiée ou non. Plusieurs méthodes de détection de falsifications ont déjà été proposées, cependant elles sont généralement incapables de détecter simultanément les différents types de falsifications et nécessitent de décoder intégralement la vidéo afin de réaliser ces détections.

Les travaux réalisés lors de mon stage ont pour objectif de mettre en place un outil permettant de détecter si une vidéo a subi une falsification à partir de caractéristiques récupérées directement depuis le bitstream de la vidéo, donc sans recourir à son décodage. Nous nous intéresserons plus particulièrement aux falsifications inter-images, à savoir la suppression, l'insertion, ou le déplacement de séquences vidéos. Pour cela, des techniques d'apprentissage automatique doivent être mises en oeuvre. Un SVM a été entraîné à l'aide d'un corpus constitué de vidéos non falsifiées (ou *pristine*) et de copies de ces vidéos ayant subi divers types de falsifications inter-images.

1.2 Presentation du laboratoire GREYC et de l'équipe SAFE

J'ai effectué mon stage dans le laboratoire de recherche en sciences du numérique GREYC (Groupe de Recherche en Informatique, Image et Instrumentation de Caen) devenu une unité mixte de recherche (UMR) CNRS depuis 2000. Le laboratoire, dirigé par Christophe Rosenberger, est sous tutelle de l'université de Caen Normandie (UNICAEN), de l'Ecole Nationale Supérieure d'Ingénieurs de Caen (ENSICAEN) ainsi que du CNRS. Le laboratoire s'étend sur 7 sites en Normandie.

L'UMR compte 180 membres dont 6 chercheurs CNRS, 23 professeurs des universités, 51 maîtres de conférences et 45 doctorants. Les domaines d'activitées du laboratoire s'organisent en 6 équipes, IMAGE, MAD, AMACC, CODAG, SAFE et ELEC. J'ai effectué mon stage au sein de l'équipe SAFE (Sécurité, Architecture, Forensique, biomEtrie), dirigée par Christophe Charrier, maître de conférence HDR à l'université de Caen Normandie. Trois professeurs des Universités, cinq maîtres de conférences (dont trois HDR), un chargé de recherche HDR au CNRS et deux ingénieurs de recherche composent cette équipe. Les thèmatiques de recherche principales de l'équipe sont la biométrie, les architectures et modèles de sécurité et enfin la science de l'investigation (forensique).

Durant mon stage, j'ai eu l'occasion d'assister à plusieurs démonstrations de projets réalisés par l'équipe comme le logiciel GREYC Keystroke, qui analyse la dynamique de frappe au clavier pour identifier l'utilisateur. J'ai aussi pu assister à une démonstration du logiciel GREYC Star, qui permet d'associer le visage de l'utilisateur à une star via des techniques d'intelligence artificielle.

Christophe Charrier et Emmanuel Giguet lui-même chargé de recherche HDR au CNRS, ont été mes encadrants durant ce stage.

1.3 Contexte du stage

Mon stage est en lien avec la thématique forensique du laboratoire, et plus particulièrement avec le projet G'DIP (GREYC Digital Investigation Platform) dans lequel les méthodes développées durant ce stage pourraient être intégrées. Le projet de la plate-forme, débuté l'année dernière, a pour objectif l'automatisation de l'analyse de traces numériques de données hétérogènes. Ces données peuvent êtres des fichiers, des textes, des images, des vidéos, des paquets réseaux. Pour les analyser, différents modules individuels doivent être implémentés. L'avantage de ce type de plate-forme est de pouvoir contenir un nombre presque illimité de fonctionnalités. On retrouve ainsi l'analyse de type de fichier, la détection de nus dans les images, la détection de fichiers dupliqués, etc.

L'intérêt d'une telle plate-forme est multiple, il peut, par exemple, être utilisé pour de l'investigation par la gendarmerie ou pour aider dans la recherche sur la protection de la vie privée.

L'objectif final des travaux réalisés durant mon stage est l'intégration d'un module consacré à la détection de falsifications vidéo dans la plate-forme. Les différents types de falsifications sont détaillés dans la section 1.4. Mon travail lui, a porté sur la détection des falsifications dites inter-images.

1.4 Falsifications des vidéos

La falsification vidéo est le fait de manipuler le contenu d'une vidéo pour en générer une nouvelle partiellement ou totalement différente. La falsification se distingue du montage vidéo, utilisé par exemple pour la réalisation de documentaires ou de films, de par l'objectif même de la modification. Là où le montage vidéo est utilisé dans le but d'améliorer l'esthétique ou la lisibilité de la vidéo, la falsification a pour but de tromper volontairement le spectateur pour influencer ses décisions.

Les falsifications vidéos peuvent être réalisées de nombreuses manières allant de la vidéo intégralement générée par des techniques d'intelligence artificielle comme les deepfakes, aux falsifications localisées à un endroit précis de la vidéo. Globalement, il est possible de départager les falsifications en trois catégories distinctes :

1. L'hypertrucage (ou deepfakes) consistant à réaliser une vidéo intégralement fausse à partir d'autres vidéos. Ces falsifications utilisent des algorithmes de deep-learning.
2. Les vidéos sur lesquelles les falsifications sont opérées sur l'intégralité d'une ou de plusieurs images. Ces falsifications sont appelées falsifications **inter-images**.
3. Les vidéos sur lesquelles les falsifications sont opérées sur une région ou un élément précis d'une ou de plusieurs images. Ces falsifications sont appelées falsifications **intra-image**s.

Les falsifications intra-images peuvent être réalisées de différentes manières, soit par copie-déplacement, soit par insertion (splicing en anglais), soit par suppression-reconstruction (inpainting en anglais).

— Le **copier-déplacer** consiste à copier une région d'une image d'une vidéo et à la dupliquer sur cette même image ou sur d'autres images.
— L'**insertion** est basée sur le même principe que le copier-déplacer à la différence que la région copiée provient d'une image d'une autre vidéo.
— Enfin, la **suppression-reconstruction** consiste à supprimer un élément d'une image puis à reconstruire les pixels manquant suite à la suppression. Un algorithme de deep-learning est généralement nécessaire pour réaliser cette dernière falsification.

Les falsifications inter-images, peuvent être classées suivant 4 types, l'insertion, la suppression, la duplication et le swap (ou inversion).

— L'**insertion** consiste à extraire un certain nombre d'images d'une vidéo B pour l'insérer dans une vidéo A.
— La **suppression** consiste à retirer un certain nombre d'images d'une vidéo.
— La **duplication** est identique à l'insertion mis à part que les images copiées proviennent de la même vidéo que celle dans laquelle elles sont insérées.
— Le **swap** consiste à intervertir l'ordre des images d'une vidéo.

La figure 1.1 présente les différents types de falsifications inter-image, l'insertion (a), la suppression (b), la duplication (c) et le swap (d).

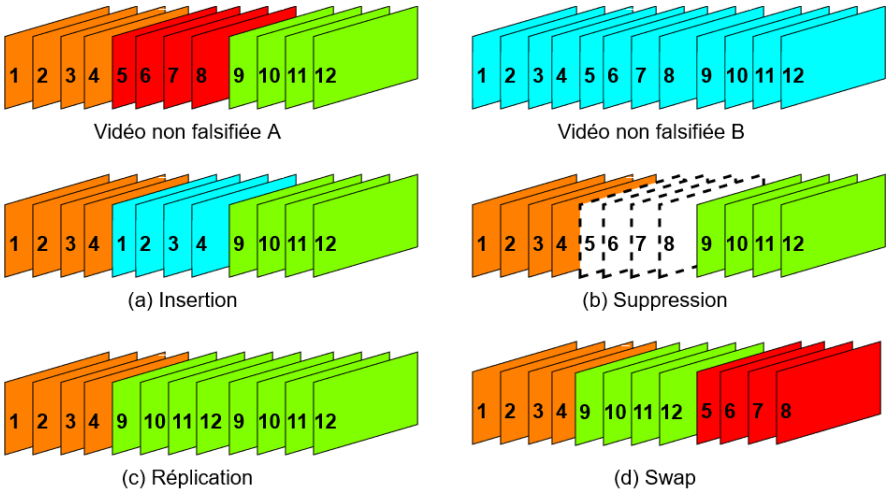

FIGURE 1.1 – Exemples de falsifications inter-images

Les traces de falsification présentes dans une vidéo dépendent du type de falsification réalisé. Par exemple, si le but d'une falsification est de supprimer le passage d'un objet dans une vidéo, une falsification intra-images pourra laisser des traces visuelles au niveau des pixels de l'image alors qu'une falsification inter-images pourra être visible par un "saut" temporel lors du passage de l'image N à l'image $N + M + 1$, M étant le nombre d'images supprimées lors de la falsification. De ce fait, il est nécessaire d'appréhender chaque catégorie de falsification de manière individuelle pour être en mesure de les détecter efficacement.

Durant ce stage, les travaux ont donc porté sur les falsifications inter-images qui sont les plus courantes étant donné la facilité avec laquelle elles peuvent être réalisées. La figure 1.2 récapitule les différentes catégories de falsification

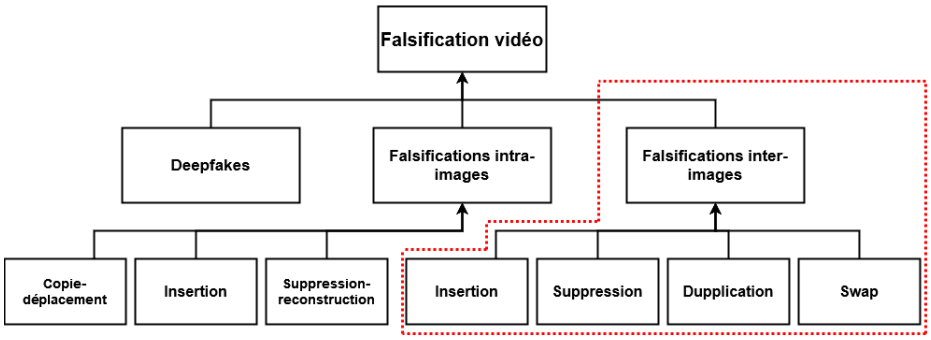

FIGURE 1.2 – Classification des méthodes de falsification vidéo

existantes et met en évidence celles étudiées durant ce stage.

Chapitre 2

Détection de manipulation de vidéos compressées

2.1 Définition et structure d'une vidéo

Une vidéo est une séquence d'images visualisée à une certaine fréquence pour donner une impression de mouvement aux spectateurs. Cette impression de mouvement est possible grâce au phénomène de la persistance rétinienne, les cellules de la rétine gardent en mémoire une image pendant un certain temps et donc, si l'enchaînement des images est suffisamment rapide, l'œil est incapable de faire la distinction entre les images successives.

Dans une vidéo, une image est décomposée en une ou plusieurs bandes (ou slice). Une bande est constituée de macroblocs les uns à la suite des autres. Ces macroblocs représentent un bloc de 16x16 pixels. Enfin, les macroblocs sont constitués de 4 blocs de luminance et 2 blocs de chrominance (la luminance et la chrominance seront définies dans la sous-section 2.2.3).

2.2 Compression MPEG-4 AVC/H264

La compression vidéo est utilisée pour réduire la taille et le débit de séquence vidéos. Pour y parvenir, on utilise des codecs (pour COdeur-DECodeur). Le fait de compresser une vidéo est généralement synonyme de pertes de données et il est donc important de trouver un équilibre entre la qualité et la taille des vidéos. Le format de compression H264-AVC est aujourd'hui le format le plus utilisé. Il a été développé conjointement par le Moving Picture Experts Group (MPEG) et le Video Coding Experts Group (VCEG) réunis sous le nom de Joint Video Team (JVT). Ce format a été publié simultanément comme la recommandation H.264 de l'ITU-T et comme la norme ISO/IEC 14496-10 : (MPEG-4 part 10) Advanced Video Coding (AVC) en 2003, mais il reste techniquement le même dans les deux cas, seul le nom change.

Le format H264 possède différents moyens pour compresser une vidéo, nous ne parlerons dans ce rapport que de deux d'entre eux car ceux-ci sont en lien direct avec les caractéristiques que nous allons utilisées plus tard (Cf. section 3.2). Les deux moyens se basent sur la redondance spatiale et temporelle présente dans l'image. À partir de ces redondances, des prédictions de macroblocs sont effectuées.

2.2.1 Redondance spatiale et temporelle

Pour réaliser la compression, le codec H264 utilise les redondances spatiale et temporelle, tel que mentionné ci-dessus.

La redondance spatiale, utilisée principalement dans les images I (Cf. section 2.2.2), se base sur le fait que dans une image, on retrouve régulièrement des macroblocs similaires voire identiques au sein d'une même image comme par exemple un ciel bleu en arrière plan. Lors de l'encodage, le codec va ainsi utiliser ce phénomène pour ne pas dupliquer les informations de macroblocs similaires. Un macrobloc encodé en se basant sur ce type de redondance est appelé macrobloc I.

La redondance temporelle, utilisée dans les images P et B (Cf. section 2.2.2), se base sur le fait que les différences entre plusieurs images successives d'une vidéo sont minimes, voire inexistantes s'il n'y a pas de mouvement. On peut alors déterminer une partie de la composition d'une image à partir des images qui la précèdent ou qui lui succèdent. A partir de cette redondance, les macroblocs ne contiendront pas toutes les informations de luminance et de chrominance, mais simplement l'emplacement du macrobloc identique présent dans l'image de référence ainsi qu'un vecteur de mouvement si l'emplacement des deux macroblocs est différent d'une image à une autre. Un macrobloc prédit à l'aide de la redondance temporelle est appelé macrobloc P si la prédiction est faite à partir de l'image précédente et macrobloc B si la prédiction est faite à partir des images précédentes et suivantes.

2.2.2 Groupe d'image (ou GOP)

Les prédictions se basant sur ces redondances spatiales et temporelles sont appelées respectivement prédiction intra-trame et prédiction inter-trame. Pour les réaliser, le codec va décomposer l'ensemble des images de la vidéo en plusieurs groupes d'images ou GOP (Group Of Pictures) pouvant être de taille variable ou fixe. Un GOP est constitué de différents types d'images :

- Une image I (ou Intracoded-frame ou image de référence) qui est une image constituée intégralement de macroblocs I. Cette image n'a donc besoin d'aucune autre information que celles qu'elle contient pour être décodée. Ce type d'image est équivalent à une image JPEG et est utilisé comme référence pour toutes les prédictions temporelles.
- Des images P (ou Predictive-frame) et des images B (bi-predictive-frame) dans lesquelles la compression est basée sur la prédiction inter-trame et intra-trame. Les images P sont prédites à partir de l'image I précédente

alors que les images B sont prédites à partir des images I et P précédentes et suivantes.

De manière générale, un GOP est toujours constitué d'une image I suivie d'images P et B ordonnées de manière fixe (Cf. figure 2.1). Cependant, le codec choisit lui-même l'agencement des GOP lors de l'encodage et il est parfois possible de trouver un nombre variable d'images B entre deux images I ou P. Il est nécessaire de ne pas créer des GOP de trop grande taille dans une vidéo pour éviter la propagation d'erreurs si une image de référence venait à être corrompue. En effet, la prédiction des images B et P étant en partie basée sur l'image I précédente, une erreur lors du décodage ou de la transmission ne pourra être corrigée que lorsque le prochain GOP sera atteint.

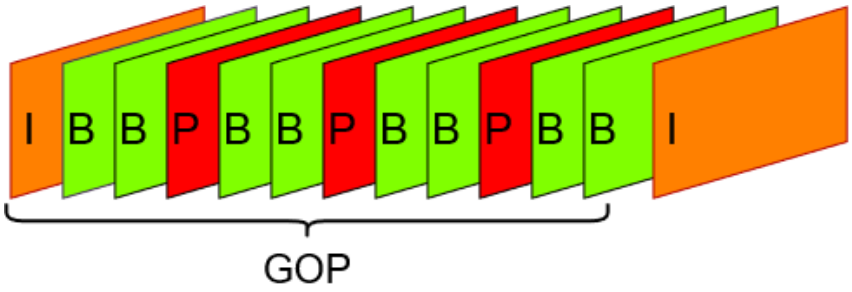

FIGURE 2.1 – Structure d'un GOP

2.2.3 Prédiction spatiale

Lors de la prédiction spatiale, un macrobloc de prédiction va être comparé au macrobloc courant C que l'on souhaite compresser. Le macrobloc de prédiction est construit à partir des macroblocs situés à coté du macrobloc C. Avec H264, il existe 13 modes de prédiction disponibles sur les composantes de luminance et 4 modes de prédiction disponibles sur les composantes de chrominance. Le principe consiste à réaliser l'ensemble des prédictions et de comparer la différence de chaque prédiction avec le macrobloc C. Le codec va ensuite séléctionner la prédiction qui minimise la différence avec C et encoder cette différence ainsi que la méthode de prédiction utilisée.

La luminance et la chrominance sont les principales informations contenues dans un macrobloc. Ce sont ces composantes qui contiennent l'information de couleur du macrobloc. Au lieu de travailler dans l'espace colorimétrique RGB, un changement d'espace couleur est opéré vers l'espace YUV. Plus spécifiquement, la luminance Y apporte l'information de la luminosité et la chrominance, présente sur les deux composantes U et V, correspond respectivement à l'axe jaune-bleu et rouge-vert. Le plus souvent, un macrobloc de taille 16x16 pixels contient l'information de la luminance sur chaque pixel et l'information de la chrominance est présente sur 1 pixel sur 4. On dit que le format de sous-échantillonage utilisé est 4 :2 :0. Il existe d'autres formats comme 4 :4 :4 (les composantes U

et V sont présentes sur chaque pixel) utilisé dans le monde du cinéma ou le format 4 :2 :2 (les composantes U et V sont présentes sur 1 pixel sur 2). Dans H264, la luminance est stockée sous forme d'une matrice de taille 16x16 et la chrominance est stockée sous forme de deux matrices de taille 8x8 chacune.

Concernant la luminance, la prédiction dans H264 peut donc être réalisée par 13 modes de prédiction différents, 4 réalisables directement sur l'entièreté du macrobloc (donc sur 16x16 pixels) et 9 autres sur des sous-blocs de taille 4x4 pixels. Parmi les différentes possibilités, le mode de prédiction verticale, horizontale, diagonale bas-gauche et à partir des valeurs moyennes sont communes aux prédictions 16x16 et 4x4. Les autres modes sont tous des prédictions diagonales, seul le sens de lecture est modifié. La figure 2.2 résume ces différents modes de prédiction.

Pour la chrominance, la prédiction est réalisée de la même manière que pour les prédictions de luminance 16x16 à la différence que les prédictions sont réalisées sur des blocs de 8x8 pixels.

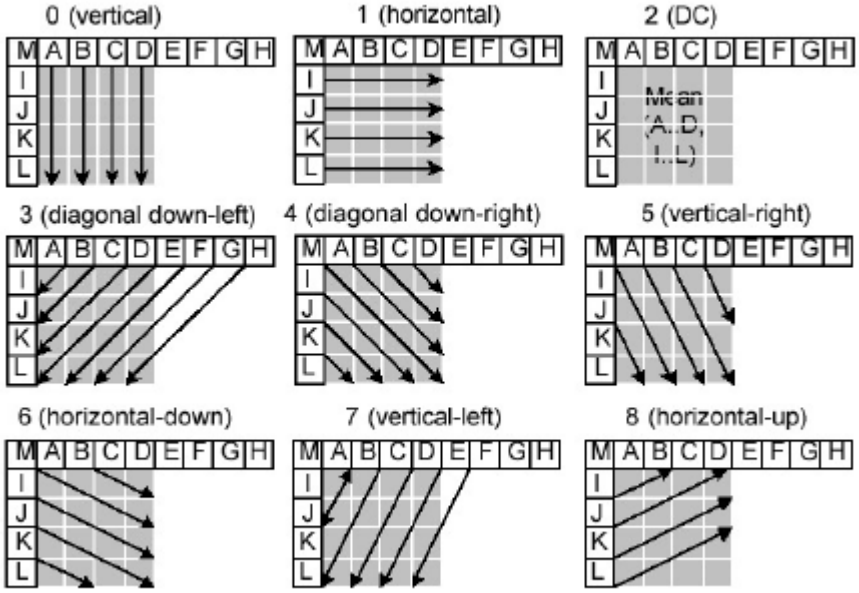

FIGURE 2.2 – Les différents modes de prédiction intra dans le format de compression H264

2.2.4 Prédiction temporelle

La prédiction temporelle se base sur le même principe que la prédiction spatiale, l'encodeur va réaliser différentes prédictions et les comparer avec le macrobloc à encoder, il selectionnera ensuite la prédiction minimisant la différence. La particularité de la prédiction temporelle est de baser les prédictions

sur les macroblocs de l'image de référence. Pour cela, il est possible dans cette méthode de partitionner les macroblocs de 4 manières distinctes : 16x16, 16x8, 8x16 et 8x8. Une des spécificités de H264 est de permettre de créer des sous-macroblocs lorsque la prédiction 8x8 est sélectionnée. Il est alors possible de choisir 4 nouvelles partitions : 8x8, 8x4, 4x8 et 4x4. La figure 2.3 résume les différents modes de partitionnement de macrobloc disponibles pour la prédiction temporelle.

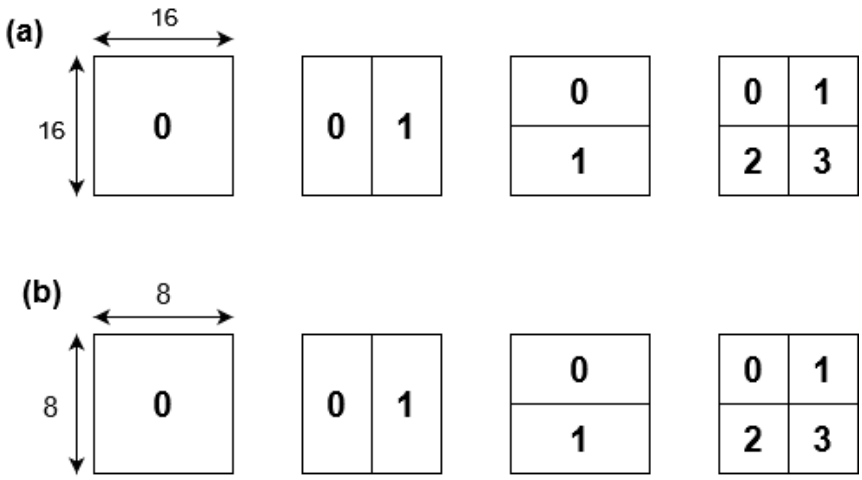

FIGURE 2.3 – Les différents modes de partitionnement de macrobloc disponibles dans le format H264 pour les macroblocs (a) et les sous-macroblocs (b)

Plus la taille choisie pour le partitionnement est petite, plus il est nécessaire de réaliser de prédiction et donc moins la compression est efficace. Il est donc important pour le codec de choisir un compromis entre la qualité à retranscrire et le taux de compression à atteindre.

A l'opération décrite ci-dessus s'ajoute un vecteur de mouvement, il va être utilisé pour chaque partition de macrobloc obtenue. Ce vecteur de mouvement correspond à la distance séparant la localisation de la partition située dans l'image de référence à la localisation de la partition que l'on souhaite prédire.

Comme pour la prédiction spatiale, le codec va finalement encoder le partitionnement utilisé, l'erreur résiduelle (la différence entre la prédiction et le macrobloc à encoder) ainsi que le vecteur de mouvement. Un macrobloc codé avec ce genre de prédiction est appelé macrobloc P s'il n'utilise qu'une image de référence (pour Predictive macrobloc) et est appelé macrobloc B (pour Bi-predictive macrobloc) s'il utilise deux images de références (une avant l'image en train d'être encodée et une après).

2.3 Méthodes de détection de falsifications de vidéos

Les méthodes de détection de falsifications sont décomposées en deux groupes, les méthodes dites "actives" et "passives" (ou aveugles).

Les méthodes actives nécessitent la présence d'informations préalables sur l'examen de la vidéo telle que la présence d'une marque (tatouage numérique et signature). Cependant, ces marques sont rarement présentes sur les vidéos et sont donc utilisées dans des cas particuliers.

A l'inverse, les méthodes passives se basent uniquement sur les caractéristiques de la vidéo et la recherche de traces de contrefaçon dans celles-ci, leur usage est donc bien plus adapté aux scénarios de la vie réelle.

Comme il a été vu dans la section 1.4, les méthodes de détection de falsifications ne peuvent généralement pas être utilisées pour détecter différentes catégories de falsification en même temps comme les deepfakes, les falsifications intra-images et les falsifications inter-images. La seule exception est la détection de compressions multiples dans un fichier vidéo (**Wang and Farid, 2006 [10]**). En effet, les vidéos sont stockées dans un format compressé pour pouvoir être stockées et diffusées plus facilement. Lors de la falsification d'une vidéo, il est d'abord nécessaire de la décompresser. Une fois les falsifications réalisées, la vidéo doit alors être recompressée une deuxième fois. Ce processus laisse des traces dans la vidéo qui peuvent être exploitées pour déterminer si elle a subi plusieurs compressions. Cependant, cette méthode échoue si l'opération de falsification intervient avant la compression ou encore lorsqu'une vidéo non-falsifiée est compressée deux fois comme cela peut être le cas pour des vidéos trouvées sur Internet.

Dans la suite de cette section, nous nous concentrerons sur les méthodes de détection de falsifications inter-images et nous dresserons un état de l'art des techniques déjà mises en place pour cette catégorie.

Concernant les méthodes actives, **Song et al. [7]** ont proposé une méthode en 2016 basée sur des signatures laissées par les logiciels d'édition vidéo. Si la signature présente dans la vidéo coïncide avec une signature enregistrée dans une base de données, alors la vidéo est considérée comme falsifiée. Cette méthode, bien que rapide, échoue lorsque la manipulation est utilisée avec un autre logiciel d'édition que ceux répertoriés. Cela rejoint le constat écrit plus haut concernant les méthodes actives dans leurs globalités.

Lin et al. [5] proposent une méthode pour détecter la duplication d'image. Celle-ci réalise un histogramme des couleurs présentes dans chaque image et vérifie la présence d'un ou plusieurs histogrammes identiques dans la vidéo. D'autres méthodes proposent la détection d'insertion, de suppression et de duplication (**Wang et al. [9]**) ou la détection d'insertion et de suppression (**Chao et al. [2]**) en se basant sur des anomalies dans le flux optique. **Sitara et al. [4]** proposent une méthode pour détecter les différents types de falsifications inter-images en se basant sur des traces de falsifications localisées dans le domaine

compressé et le domaine spatio-temporel. De manière générale, les techniques de détection se basant sur des informations disponibles dans le domaine spatio-temporel sont efficaces et permettent de détecter et même différencier plusieurs types de falsifications en même temps, cependant un inconvénient majeur de ces méthodes subsiste : il est obligatoire de décoder entièrement la vidéo avant de réaliser les vérifications. Cela peut être problématique si la vérification doit être réalisée rapidement et sur un très grand nombre de vidéos, par exemple la vérification de l'intégralité des bandes des caméras de vidéosurveillance d'un bâtiment ou d'une ville.

D'autres méthodes existent et se basent uniquement sur la détection de traces présentes dans le domaine compressé de la vidéo, en exemple le travail publié par **Gironi et al. [3]**. Pour détecter une suppression ou une insertion dans une vidéo, la méthode se base sur la variation des types de prédiction utilisés dans les images P. **Su et al. [8]** utilisent la présence d'artefacts périodiques dans les coefficients des transformés en cosinus discretes. Ces artefacts apparaissent lors d'une recompression d'images P et B. Ils sont dûs aux changements de place possible des images après une insertion ou une suppression. Ces deux méthodes sont efficaces lorsque la taille des GOP est fixe, cependant elles échouent si la falsification est opérée sur un ou plusieurs GOPs entiers.

Enfin, **Sitara et al. [6]** proposent un article résumant d'autres méthodes de détections de falsification inter-image, intra-images et de compressions multiples.

La méthode présentée dans ce rapport est basée sur la récupération d'informations localisées dans le domaine compressé de la vidéo et permet de détecter chaque type de falsifications inter-images avec une performance plus ou moins bonne.

Chapitre 3

Méthodes de détection implémentées

Lors de mon stage, trois étapes principales ont été nécessaires pour l'implémentation des différentes méthodes de détection de falsification vidéo, à savoir :

1. La création d'un corpus de vidéos falsifiées/non-falsifiées pour entraîner nos classifieurs.
2. La récupération de caractéristiques des vidéos directement depuis le bitstream.
3. L'entraînement et la comparaison de classifieurs sur le corpus créé.

La figure 3.1 décrit ce procédé en détaillant les tâches à réaliser au cours de chacune des étapes.

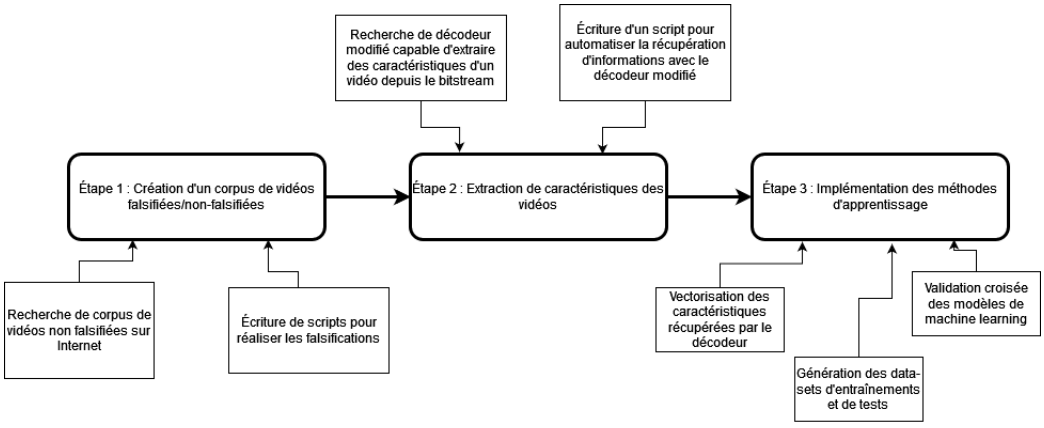

FIGURE 3.1 – Schéma illustrant les différentes étapes à réaliser

3.1 Création du corpus

3.1.1 Recherche de corpus disponible

La création du corpus a été réalisée en deux temps, tout d'abord une recherche dans les différents travaux déjà publiés pour vérifier s'il existe un ou des corpus de vidéos falsifiées/non-falsifiées déjà constitués et disponibles en libre accès. En réalité, le nombre de corpus disponible pour la détection de falsifications de vidéos est très limité. Nous avons donc décidé de réaliser notre propre corpus de vidéos à partir de vidéos non-falsifiées, ces dernières étant faciles à récupérer.

Au total, 111 vidéos ont été récupérées provenant de 2 corpus différents. Le premier corpus provient du projet REWIND (REVerse engineering of audio-VIsual coNtent Data) **[1]**et contient 10 vidéos *.mp4* compressées en utilisant le format H264 avec une résolution de 320x240 pixels. Le deuxième corpus provient du projet VIFFD (Video Inter-Frame Forgeries Detection) et contient 101 vidéos *.mp4* compressées en utilisant le format H264 avec 15 vidéos possédant une résolution de 1920x1080 pixels et 86 vidéos avec une résolution de 720x404 pixels. Chaque vidéo possède un framerate de 25 images par seconde. Le corpus contenait aussi des vidéos avec des falsifications de type inter-images, cependant, elles n'étaient pas réalisées pour chacune des vidéos et ne concernaient pas tous les types de falsifications, c'est pourquoi nous avons décidé de ne pas les récupérer et à la place, de réaliser nos propres falsifications.

3.1.2 Écriture des scripts pour créer les vidéos falsifiées

Après récupération des vidéos non-falsifiées, il a été nécessaire de créer les vidéos avec falsifications. Pour cela, le software *ffmpeg* a été utilisé et plusieurs scripts *shell* écrits. Un premier script recompresse toutes les vidéos originales à l'aide du codec libx264 et fixe la taille des GOPs à 25 images (ce qui permet de faire coïncider la taille des GOPs avec le framerate des vidéos et simplifie ainsi l'utilisation de ffmpeg). Ensuite, un second script est utilisé pour réaliser les vidéos falsifiées. Ce script créé, pour chaque vidéo originale, quatre vidéos falsifiées, une pour chaque type de falsification. L'insertion est réalisée en récupérant 2 secondes, soit 50 images, d'une vidéo choisie aléatoirement dans le corpus et en les insérant après 3 secondes de la vidéo originale. Pour la suppression, les images 50 à 100 sont retirées. Pour la copie, les images 25 à 75 sont copiées et insérées après les 50 premières images de la vidéo. Enfin, le swap est réalisé en intervertissant les images 50 à 75 et 75 à 100.

Au total, après les opérations de falsifications, le corpus contient **555** vidéos avec **111** vidéos non-falsifiées (ou *pristine* en anglais) et 444 vidéos falsifiées réparties selon le type de falsifications, **111** insertions, **111** suppressions, **111** duplications et **111** swaps. Les 2 diagrammes de la figure 3.2 récapitulent la répartition des vidéos du corpus. Le premier montre la répartition des vidéos en fonction des types de falsification. Le deuxième diagramme renseigne sur le

nombre de vidéos non-falsifiées en fonction de leur durée.

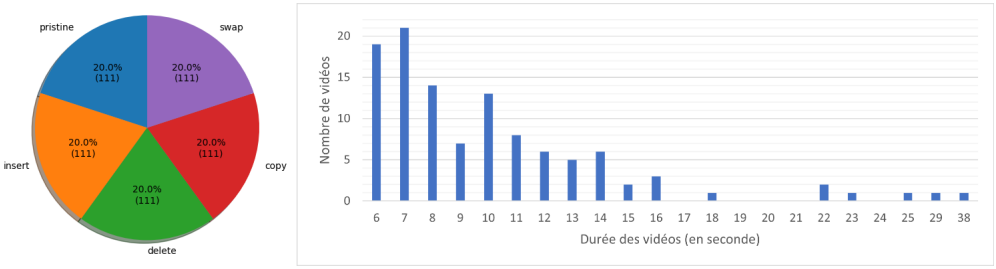

FIGURE 3.2 – Répartition des vidéos dans le corpus créé

3.2 Extraction des caractéristiques des vidéos

Pour extraire des caractéristiques d'une vidéo, il est d'abord nécessaire d'en décoder une partie. Nous avons donc dans un premier temps recherché un codec H264 écrit en python sur Github. Malheureusement, un codec a pour objectif d'être rapide lors du décodage et l'utilisation d'un langage interprété comme le Python semble contradictoire avec ce but. De ce fait, peu de personnes ont travaillé sur la réalisation d'un codec écrit en Python. Le seul décodeur trouvé n'avait qu'un but pédagogique et n'était capable de décoder qu'un profil de vidéo précis. Cependant, l'étude de ce décodeur à tout de même permis d'apprendre plus explicitement le fonctionnement du décodeur et la compression H264.

Finalement, le langage Python a été abandonné pour l'extraction et le choix s'est porté sur une version modifiée du codec H264 de référence (JM 17.1). Ce codec est écrit en langage C. Le décodeur modifié a été récupéré et modifié par deux autres membres du laboratoire, Alexandre Ninassi et Hugo Merly, lors de la réalisation d'un projet antérieur. Il permet de récupérer pour chaque image d'une vidéo, 27 caractéristiques (ou *features*) depuis le bitstream. Ces caractéristiques sont les suivantes :

- f_1 : le bitrate qui exprime la quantité de données transmises par seconde. Un bitrate élevé correspond à plus d'information stockée dans l'image.
- f_2, f_3 : la valeur moyenne du pas de quantification (QP) ainsi que le delta QP.
- f_4, f_5 : la valeur moyenne et maximale de la longueur des vecteurs de mouvement dans l'image.
- f_6, f_7 : la valeur moyenne et maximale de la longueur d'erreur des vecteurs de mouvement.
- f_8, f_9, f_{10} : Pourcentages correspondants au codage des macroblocs utilisés dans l'image : intra-codé (prédiction intra), inter-codé (prédiction inter) ou skip (pas de donnée codée dans le macrobloc).
- $f_{11}, ..., f_{13}$: Pourcentages des macroblocs I selon leur mode de partitionnement. Ces modes sont 16x16, 8x8 et 4x4.

— $f_{14}, ..., f_{17}$: Pourcentages des macroblocs P selon leur mode de partitionnement. Ces modes sont 16x16, 16x8, 8x16 et 8x8.
— $f_{18}, ..., f_{20}$: Pourcentages des sous-macroblocs P selon leur mode de partitionnement. Ces modes sont 8x4, 4x8 et 4x4.
— $f_{21}, ..., f_{24}$: Pourcentages des macroblocs B selon leur mode de partitionnement. Ces modes sont 16x16, 16x8, 8x16.
— $f_{25}, ..., f_{27}$: Pourcentages des macroblocs P et B de type *skip* et des macroblocs B de type *direct*. Ces macroblocs ne sont pas codés. Le décodeur s'occupe de déterminer leur valeurs en fonction des macroblocs voisins (mode *skip*) ou du macrobloc situé à la même position dans l'image de référence (mode *direct*).

Dans le format H264, le bitstream possède une structure décrite dans l'annexe B de la recommandation de l'ITU-T. La figure 3.3 présente la structure de ce bitstream.

Les caractéristiques sont récupérées à deux niveaux différents du bitstream. Le bitrate est récupéré dans l'en-tête de la bande alors que les autres caractéristiques sont calculées à partir des informations contenues dans les données du macrobloc.

Le décodeur modifié prend en entrée un fichier de configuration .cfg contenant le chemin de la vidéo à décoder et génère en sortie un fichier .csv contenant pour chaque image de la vidéo le type d'image (I, P, B) ainsi que la valeur des 27 caractéristiques. Un script python a été écrit pour automatiser la récupération des caractéristiques sur les 555 vidéos créées auparavant.

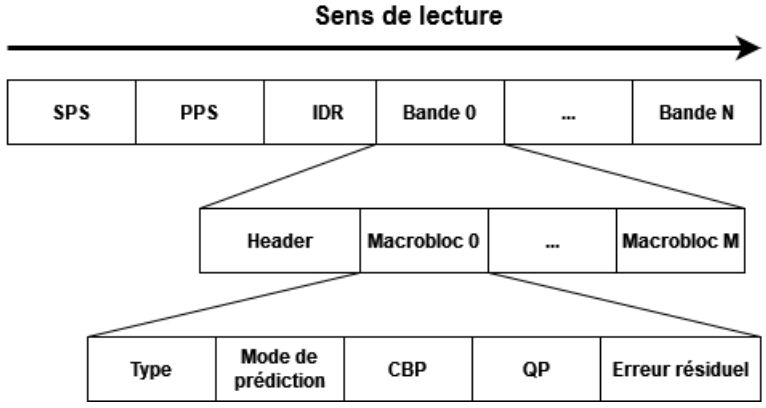

FIGURE 3.3 – Structure du bitstream pour le format H264

3.3 Mise en place des algorithmes d'apprentissage automatique

La mise en place des algorithmes d'apprentissage automatique (ou machine learning) peut être divisée en 3 étapes. La première est la transformation des

données extraites par le décodeur en vecteurs utilisables par un classifieur, ce processus est appelé vectorisation. La seconde étape est la création, à partir des vecteurs récupérés précédemment, d'un dataset d'entraînement et d'un dataset de test. Enfin, la dernière étape consiste à implémenter le code des classifieurs et vérifier leurs performances. On utilise pour cela la méthode de cross-validation.

Pour réaliser l'intégralité des tâches, j'ai écrit un programme python composé de 2 classes :

1. La classe **Parameters** associée aux étapes 1 et 2, récupérant l'ensemble des données concernant les vidéos et réalisant la vectorisation et la création des datasets de test et d'entraînement. La classe possède aussi une méthode pour écrire une synthèse d'une simulation au format Json. Ces synthèses contiennent les informations sur le corpus utilisé, comme le nombre de vidéos et la répartition de celles-ci, les caractéristiques utilisées pour construire les vecteurs ainsi que les résultats de la simulation.
2. La classe **Classifier** associée à l'étape 3, permet de réaliser des simulations sur divers classifieurs et de calculer les indices de performance choisis à chaque simulation. Ces indices sont au nombre de 4 et sont l'exactitude, la précision, le rappel et le F-score. Ces indices sont détaillés dans la sous-sections 3.3.4

En plus de ces deux classes, j'ai écrit un script python pour réaliser une synthèse de toutes les simulations réalisées au format HTML à partir des fichiers Json générés à chaque simulation et pour chaque classifieur.

3.3.1 Vectorisation des caractéristiques récupérées

Lors de la phase de vectorisation, deux manières de procédés ont été envisagées :

1. La première, plus basique, consiste à récupérer la valeur moyenne de chaque caractéristique sur l'ensemble de la vidéo et de construire les vecteurs à partir de ces moyennes.
2. La deuxième approche consiste à tenir compte des GOPs, ainsi les valeurs des caractéristiques de chaque GOP de la vidéo sont calculées de la façon suivante :

$$Val_{GOP} = \begin{cases} \frac{1}{3}\left(f_{I,k} + \frac{1}{n_P}\sum_{i=1}^{n_P} f_{P_i,k} + \frac{1}{n_B}\sum_{j=1}^{n_B} f_{B_j,k}\right) & si\, n_P\, et\, n_B \neq 0 \\ \frac{1}{2}\left(f_{I,k} + \frac{1}{n_B}\sum_{j=1}^{n_B} f_{B_j,k}\right) & si\, n_P\, = 0\, et\, n_B \neq 0 \\ \frac{1}{2}\left(f_{I,k} + \frac{1}{n_P}\sum_{i=1}^{n_P} f_{P_i,k}\right) & si\, n_P\, \neq 0\, et\, n_B = 0 \\ f_{I,k} & si\, n_P\, et\, n_B = 0 \end{cases}, \forall\, k \in [1, ..., 27]$$

avec n_P, n_B : le nombre de frame P et B dans le GOP
$f_{I,k}$: valeur de la k-ième features pour la frame I du GOP

$f_{P_i,k}$, $f_{B_j,k}$: valeur de la k-ième features pour la frame i-ème (resp j-ième) frame P (resp B) du GOP

A ce stade, les composantes d'un vecteur sont construites en récupérant la moyenne, l'écart-type, le minimum et le maximum de chaque caractéristique calculée.

L'avantage de cette technique est de récupérer une valeur minimum et maximum exploitable pour la détection de falsification. Si elles sont récupérées avec la première technique, ces valeurs sont biaisées à cause des différents types d'image. Par exemple, une image I ne peut réaliser que de la prédiction intra-image et donc la valeur de la feature f_8 associée au taux d'intra-prédiction sera forcément égale à 1 et les features f_9 et f_{10} seront égales à 0. Cette observation se reproduit pour d'autres features. Il a donc été décidé d'utiliser la deuxième méthode pour construire nos vecteurs.

Les vecteurs sont créés à l'aide de la fonction **vectors_comp()**. Cette fonction parcourt les fichiers csv générés par le décodeur et génère un dictionnaire avec comme clé le nom de la vidéo (*Video_X* pour les vidéos non-falsifiées et *Insert_X, Delete_X, Copy_X* et *Swap_X* pour les vidéos falsifiées) et comme valeurs associées à ces clés, le vecteur construit comme décrit ci-dessus.

3.3.2 Création des datasets d'entrainement et de test

Après la vectorisation des données effectuée, il est nécessaire de pouvoir séparer les vecteurs en deux datasets, un pour l'entrainement et un pour le test. En apprentissage supervisé, on décompose très souvent son corpus en deux datasets pour vérifier la bonne généralisation du classifieur. Le dataset d'entraînement sert, comme son nom l'indique à entraîner le classifieur. Les performances d'un algorithme d'apprentissage automatique dépendent directement des données qui sont utilisées pour l'entrainement, il est donc important d'en avoir suffisament et que celles-ci soient généralisables.

Les données présentent pour le test sont des données jamais rencontrées par le classifieur et qui permettent ainsi de vérifier qu'il ne fait pas de sur-entraînement (un classifieur sur-entraîner est trop adapté aux données d'entraînements et perd alors la capacité de bien classer de nouvelles données).

Les datasets sont créés à l'aide de la fonction **creation_train_and_test_dataset(vectors_dictionnary, test_size)** avec vectors_dictionnary le dictionnaire généré précédemment et test_size un float compris entre 0.0 et 1.0 correspondant à la proportion des données du corpus à inclure dans le dataset de test. Les datasets sont construits de la manière suivante :

1. Sélection aléatoire de test_size*nb_vidéos_non_falsifiées parmi les vecteurs des vidéos non falsifiées.
2. Récupération des vecteurs des vidéos falsifiées associés aux vidéos non-falsifiées. Par exemple, si le vecteur de la vidéo Video_0 est sélectionné,

alors les vecteurs des vidéos Insert_0, Delete_0, Copy_0 et Swap_0 seront sélectionnés en fonction des types de falsification choisis pour les simulations.

3. Les vecteurs restants sont utilisés pour construire le dataset d'entraînement.

Enfin, à chaque fois qu'un vecteur est récupéré, sa catégorie (falsifiée/non-falsifiée) est récupérée en même temps. Ainsi, la fonction retourne une liste contenant :

— X_train : liste des vecteurs utilisés pour la phase d'entraînement.
— y_train : liste des catégories associées aux vecteurs utilisés pour la phase d'entraînement.
— X_test : liste des vecteurs utilisés pour la phase de test.
— y_test : liste des catégories associées aux vecteurs utilisés pour la phase de test.

3.3.3 Implémentation et validation des classifieurs

Suite à la construction des vecteurs, la dernière étape est l'implémentation puis la validation des classifieurs. L'implémentation d'un classifieur de type SVM (Support Vector Machine) était prévue. En fait, d'autres classifieurs ont été implémentés pour avoir des moyens de comparaison. Tous les classifieurs ont été implémentés en python à partir de la librairie Scikit-Learn, une librairie spécialisée dans le machine learning. Au total, 8 algorithmes de classification sont présents à savoir :

— Un arbre de décision (Desision Tree)
— Une classification en utilisant la méthode des k plus proches voisins (k-nearest neighbors)
— Une régression logistique (Logistic Regression)
— Une classification naïve bayésienne (Naive Bayes method)
— Un SVM avec un noyau linéaire avec et sans pondération des classes de données (Support Vector Machine with linear kernel et Support Vector Machine with linear kernel and class weight)
— Un SVM avec un noyau rbf (Radial Basis Function) avec et sans pondération des classes de données (Support Vector Machine with rbf kernel et Support Vector Machine with rbf kernel and class weight)

3.3.4 Les indices de performance choisis

Pour l'évaluation de la performance des classifieurs, quatre indices ont été retenus, à savoir l'exactitude, la précision, le rappel et le F-score. Dans notre cas, **l'exactitude** (a) représente le nombre de vidéos bien catégorisées rapporté à

l'ensemble des vidéos à classer. **La précision** (b) représente le nombre de vidéos falsifiées bien catégorisées rapporté au nombre de vidéos classées comme étant falsifiées. **Le rappel** (c) représente le nombre de vidéos falsifiées bien catégorisées rapporté à l'ensemble des vidéos falsifiées présentes dans le corpus. Enfin, **le F-score** est la moyenne harmonique de la précision et du rappel, calculée en utilisant la formule suivante :

$$F\text{-}score = 2\,\frac{précision\,.\,rappel}{précision + rappel}$$

La figure 3.4 propose une illustration des notions d'exactitude, de précision et de rappel.

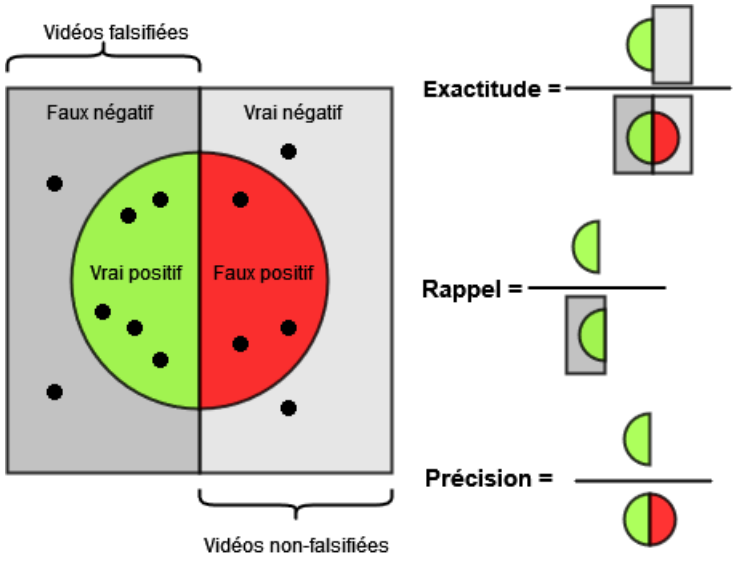

FIGURE 3.4 – Illustration des notions d'exactitude, de précision et de rappel

Chapitre 4

Analyse des résultats

Pour tester les algorithmes de classification, 999 simulations ont été réalisées pour chaque classifieur avec un dataset de test et d'entraînement généré à chaque simulation par la fonction **creation_train_and_test_dataset(vectors_dictionnary, test_size)**. La répartition des vidéos dans ces datasets ainsi générés est de 80% des données pour l'entrainement et les 20% restantes pour la phase de test. De plus, quatre séries de simulations supplémentaires ont été réalisées sur des sous-corpus pour évaluer les performances. Le corpus principal et les 4 sous-corpus sont constitués de la manière suivante :

- Corpus principal : toutes les vidéos non-falsifiées et toutes les vidéos falsifiées (insertion, suppression, duplication et swap), taille du corpus : 555 vidéos
- Sous-corpus 1 : toutes les vidéos non-falsifiées et toutes les vidéos falsifiées de type insertion, taille du corpus : 222 vidéos
- Sous-corpus 2 : toutes les vidéos non-falsifiées et toutes les vidéos falsifiées de type suppression, taille du corpus : 222 vidéos
- Sous-corpus 3 : toutes les vidéos non-falsifiées et toutes les vidéos falsifiées de type duplication, taille du corpus : 222 vidéos
- Sous-corpus 4 : toutes les vidéos non-falsifiées et toutes les vidéos falsifiées de type swap, taille du corpus : 222 vidéos

La durée totale des simulations (comprenant la construction des datasets, la phase d'entraînement et de test) varie en fonction des simulations et est de 2h53min pour le corpus principal, 41min pour le sous-corpus 1, 57min pour le sous-corpus 2, 1h23min pour le sous-corpus 3 et 1h24min pour le sous-corpus 4.

Après les 999 simulations réalisées pour l'ensemble des corpus, le script **html_report.py** permet de réaliser une synthèse au format HTML, on retrouve dans cette synthèse les informations relatives à la répartition du corpus comme dans la figure 3.2, la liste des features utilisées ainsi que le tableau comparatif de la valeur moyenne des indices de performance utilisés sur les 999 simulations pour chaque classifieur. Une autre comparaison avec des tableaux

Classifieur	Exactitude	Précision	Rappel	F-score
Arbre de décision	0.78	0.89	0.82	0.85
k plus proches voisins	0.79	0.81	**0.96**	0.88
Régression logistique	0.89	0.93	0.94	0.93
Naives Bayes	0.62	0.84	0.65	0.73
SVM linéaire	**0.90**	0.94	0.93	**0.94**
SVM linéaire avec pondération	0.88	**0.96**	0.89	0.92
SVM rbf	0.89	0.93	0.93	0.93
SVM rbf avec pondération	0.86	0.95	0.88	0.91

TABLE 4.1 – Tableau comparatif des valeurs moyennes de chaque métriques pour les 8 classifieurs après simulations sur le corpus principal

similaires est disponible, présentant les valeurs des indices pour les modèles ayant obtenus les pires et les meilleurs taux d'exactitude ainsi que les matrices de confusion associées .

Concernant les sous-corpus 1,2,3 et 4, les résultats pour les SVMs avec pondération des classes ne sont pas inclus étant donné que les sous-corpus sont équilibrés (111 vidéos non-falsifiées pour 111 vidéos falsifiées).

4.1 Corpus principal : toutes les falsifications

Les résultats obtenus à l'issue des 999 simulations sur le corpus principal sont présentés dans le tableau 4.1. Les résultats semblent prometteurs pour 5 modèles de classification, à savoir les SVMs avec noyau linéaire avec et sans pondération des classes qui atteignent respectivement 90% et 88% d'exactitude. Les SVMs avec noyau rbf avec et sans pondération des classes qui atteignent respectivement 89% et 86% d'exactitude ainsi que la régression logistique qui atteint 89% d'exactitude. Les autres classifieurs atteignent des performances plus mitigées comprises entre 62% pour la naïve bayésienne et 79% pour la méthode des k plus proches voisins.

Concernant les SVM, le fait de pondérer les classes ne semble pas améliorer les performances malgré le fait que le corpus soit déséquilibré (111 vidéos non falsifiées pour 444 vidéos non-falsifiées). L'exactitude décroît ainsi d'environ 2% en moyenne avec la pondération. De manière générale, le SVM avec noyau linéaire semble être l'algorithme d'apprentissage le plus efficace pour classer les vidéos du corpus.

Les valeurs moyennes de précision et de rappel semblent elles aussi encourageantes avec des résultats supérieurs à 90% pour les trois modèles obtenant les meilleurs scores d'exactitude (SVM avec noyau linéaire, SVM avec noyau rbf et régression logistique). Cela se traduit, pour la précision que la grande majorité des vidéos falsifiées sont bien classées et que, pour le rappel, peu de vidéos falsifiées sont classées comme étant non-falsifiées.

4.2 Sous-corpus 1 : falsifications de type insertion uniquement

Pour le sous-corpus 1, la répartition des vidéos dans les différents datasets est de 178 vidéos pour l'entraînement, réparties en 89 vidéos non-falsifiées et 89 vidéos avec insertion. Pour la phase de test, 44 vidéos réparties de la même manière sont utilisées. Le tableau 4.2 présente les résultats obtenus à l'issue des 999 simulations .

Les résultats sur ces essais sont plus satisfaisants que précédemment et les vidéos falsifiées présentant des insertions semblent être facilement séparables des vidéos non-falsifiées. Ainsi, tous les classifieurs obtiennent avec ce corpus une valeur moyenne d'exactitude supérieure ou égale à 86% sur les 999 simulations. Une fois encore, les SVMs avec noyau linéaire et rbf ainsi que la Régression logistique obtiennent les meilleures performances avec respectivement 91%, 90% et 92% d'exactitude, cela se traduit de manière concrète par le fait qu'environ 4 vidéos seulement sont mal classées sur les 44 initiales.

De la même manière, la valeur de précision est globalement très bonne pour l'entièreté des classifieurs avec un score supérieur à 90%.

Enfin, le score de rappel reste bon pour les SVMs et la Régression logistique ($\geq$90%) mais diminue plus ou moins fortement pour les autres classifieurs, cela se traduit par une proportion plus élevée de vidéos falsifiées mal classées.

Classifieur	Exactitude	Précision	Rappel	F-score
Arbre de décision	0.89	0.90	0.89	0.89
k plus proche voisin	0.86	0.94	0.77	0.85
Régression logistique	**0.92**	**0.94**	0.90	**0.92**
Naives Bayes	0.87	1.00	0.75	0.85
SVM linéaire	0.91	0.91	**0.92**	0.91
SVM rbf	0.90	0.91	0.91	0.90

TABLE 4.2 – Tableau comparatif des valeurs moyennes de chaque métriques pour les 6 classifieurs après simulations sur le sous-corpus 1

4.3 Sous-corpus 2, 3 et 4

Pour les sous-corpus 2, 3 et 4, la répartition des vidéos dans les différents datasets est réalisée de la même manière que précédemment avec 178 vidéos non-falsifiées et 178 vidéos falsifiées dans le dataset d'entraînement. La différence entre chaque sous-corpus se situe dans les types de vidéos falsifiées utilisés qui sont respectivement les vidéos avec suppression, les vidéos avec duplication et enfin les vidéos avec swap. Pour la phase de test, 44 vidéos non-falsifiées et 44 vidéos falsifiées sont utilisées. Les résultats obtenus à l'issue des 999 simulations sont présentés dans le tableau 4.3 pour le sous-corpus 2 (vidéos avec suppression), dans le tableau 4.4 pour le sous-corpus 3 (vidéos avec duplication) et dans

le tableau 4.5 pour le sous-corpus 4 (vidéos avec swap).

Les résultats obtenus avec ces essais sont similaires quel que soit le type de falsification choisi. Les résultats sont globalement moins satisfaisants que pour le sous-corpus 1 utilisant les falsifications de type insertion. On obtient de maigres scores d'exactitude pour les modèles utilisant les arbres de décision (respectivement 73%, 72% et 75%), la méthode des k plus proches voisins (respectivement 61%, 57% et 56%) et la classification naïve bayésienne (respectivement 65%, 54% et 53%). A l'inverse, les résultats obtenus avec les autres classifieurs sont plus encourageants avec des valeurs d'exactitudes comprises aux alentours de 83%. Seul le SVM avec noyau rbf obtient des résultats inférieurs lors de la classification des vidéos avec duplication (78%).

Comme pour les valeurs d'exactitudes, les autres indices de performance calculés pour ces sous-corpus sont inférieurs comparés à précédemment. Ainsi, aucun des classifieurs n'obtient de résultat supérieur à 86%.

Tout ces résultats permettent de mettre en évidence une difficulté à détecter les suppressions, les duplications ainsi que les swaps en utilisant la méthode présentée. Cependant, au vu des scores on peut considérer que les prédictions sont assez fiables pour 3 classifieurs.

Classifieur	**Exactitude**	**Précision**	**Rappel**	**F-score**
Arbre de décision	0.73	0.74	0.72	0.73
k plus proche voisin	0.61	0.61	0.62	0.61
Régression logistique	**0.85**	**0.85**	**0.86**	**0.85**
Naives Bayes	0.65	0.64	0.71	0.67
SVM linéaire	0.84	0.84	0.85	0.84
SVM rbf	0.83	0.82	0.84	0.83

TABLE 4.3 – Tableau comparatif des valeurs moyennes de chaque métriques pour les 6 classifieurs après simulations sur le sous-corpus 2

Classifieur	**Exactitude**	**Précision**	**Rappel**	**F-score**
Arbre de décision	0.72	0.74	0.71	0.72
k plus proche voisin	0.57	0.57	0.61	0.58
Régression logistique	0.82	0.81	0.83	0.82
Naives Bayes	0.54	0.52	0.80	0.63
SVM linéaire	**0.83**	**0.82**	**0.84**	**0.83**
SVM rbf	0.78	0.78	0.80	0.78

TABLE 4.4 – Tableau comparatif des valeurs moyennes de chaque métriques pour les 6 classifieurs après simulations sur le sous-corpus 3

Classifieur	Exactitude	Précision	Rappel	F-score
Arbre de décision	0.75	0.75	0.76	0.75
k plus proche voisin	0.56	0.56	0.59	0.57
Régression logistique	0.84	0.83	**0.86**	0.84
Naives Bayes	0.53	0.52	0.69	0.59
SVM linéaire	**0.86**	**0.86**	**0.86**	**0.86**
SVM rbf	0.81	0.81	0.82	0.81

TABLE 4.5 – Tableau comparatif des valeurs moyennes de chaque métriques pour les 6 classifieurs après simulations sur le sous-corpus 4

4.4 Discussions

De manière globale, le SVM linéaire et la régression logistique obtiennent les meilleurs résultats sur les différents tests. Ils semblent être en mesure de déterminer avec une bonne probabilité si une vidéo a subi une falsification inter-image. Il est aussi intéressant de noter que malgré ces diminutions, les résultats obtenus avec les falsifications de type swap sont similaires aux résultats obtenus dans d'autres travaux (**[4]**).

Cependant, quelques points peuvent être soulevés. Tout d'abord, notre corpus est assez limité en nombre de vidéos. Il serait intéressant d'en récupérer un plus grand nombre provenant de plus nombreuses sources. D'autre part, lors de la création des vidéos falsifiées, tous les paramètres de ffmpeg sont restés identiques. Une extension du corpus avec de nouvelles vidéos et en faisant varier certains paramètres comme la taille des GOP, le pas de quantification ou encore l'emplacement des falsifications (actuellement sur des GOPs entier), pourrait permettre d'obtenir des résultats plus représentatifs sur les performances réelles de notre méthode.

Conclusion

Les travaux réalisés durant ce stage ont eu pour objectif de réaliser l'implémentation d'un classifieur binaire pour déterminer si une vidéo est falsifiée ou non. Pour cela, un corpus de vidéos falsifiées/non-falsifiées a d'abord dû être créé. Un décodeur modifié a ensuite été utilisé pour extraire les caractéristiques de ces vidéos depuis le bitstream. Enfin, des algorithmes d'apprentissage ont été mis en place et testés sur le corpus.

Finalement, les résultats montrent une bonne capacité pour notre méthode à classifier les vidéos. On obtient ainsi un taux d'exactitude de 90% en utilisant un SVM linéaire. Ce résultat diminue lorsqu'on compare les vidéos non-falsifiées à chaque type de falsification individuellement. On obtient, pour un SVM linéaire, un taux d'exactitude de 84% pour les suppressions, 83% pour les duplications et 86% pour les swaps.

Une perspective possible de ces travaux est la mise en place d'un classifieur multi-classes afin de détecter précisément le type de falsification utilisé. Concernant les limites de notre méthode, comme expliqué dans l'introduction, la falsification et le montage vidéo ont le même principe technique, il semble donc plus difficile d'utiliser notre méthode pour détecter si un film a été falsifié.

Durant ce stage, j'ai eu l'opportunité de réaliser et d'assister aux différentes facettes que constituent le travail de chercheur : l'appréhension d'articles scientifiques, l'implémentation de solutions encore jamais renseignées mais aussi la vie au sein d'un laboratoire de recherche.

Le travail réalisé m'a permis d'acquérir et d'approfondir mes connaissances sur certains sujets. Tout d'abord, dans le domaine de la compression H264, ce travail fût peu aisé au départ du à la complexité et l'étendue de ce domaine. J'ai été curieux de découvrir la falsification vidéo ainsi que la détection de falsification vidéo, plus intuitif à appréhender que la compression. Parallèlement, ce stage m'a permis de développer et d'enrichir mes compétences de programmation dans le langage Python. dans l'utilisation de librairie ou software comme Scikit-learn ou ffmpeg.

Bibliographie

[1] Paolo Bestagini, Simone Milani, Marco Tagliasacchi, and Stefano Tubaro. Local tampering detection in video sequences. In *2013 IEEE 15th International Workshop on Multimedia Signal Processing (MMSP)*, pages 488–493, 2013.

[2] Juan Chao, Xinghao Jiang, and Tanfeng Sun. A novel video inter-frame forgery model detection scheme based on optical flow consistency. pages 267–281, 10 2012.

[3] A. Gironi, M. Fontani, T. Bianchi, A. Piva, and M. Barni. A video forensic technique for detecting frame deletion and insertion. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6226–6230, 2014.

[4] Sitara Kk and Babu Mehtre. Detection of inter-frame forgeries in digital videos. *Forensic Science International*, 289, 05 2018.

[5] Guo-Shiang Lin, Jie-Fan Chang, and Cheng-Hung Chuang. Detecting frame duplication based on spatial and temporal analyses. In *2011 6th International Conference on Computer Science Education (ICCSE)*, pages 1396–1399, 2011.

[6] K. Sitara and B.M. Mehtre. Digital video tampering detection : An overview of passive techniques. *Digital Investigation*, 18 :8–22, 2016.

[7] Jieun Song, Kiryong Lee, Wan Yeon Lee, and Heejo Lee. Integrity verification of the ordered data structures in manipulated video content. *Digital Investigation*, 18 :1–7, 2016.

[8] Yuting Su, Weizhi Nie, and Chengqian Zhang. A frame tampering detection algorithm for mpeg videos. In *2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference*, volume 2, pages 461–464, 2011.

[9] Wan Wang, Xinghao Jiang, Shilin Wang, Meng Wan, and Tanfeng Sun. Identifying video forgery process using optical flow. pages 244–257, 07 2014.

[10] Weihong Wang and Hany Farid. Exposing digital forgeries in video by detecting double mpeg compression. In *Proceedings of the 8th Workshop on Multimedia and Security*, MM&Sec '06, page 37–47, New York, NY, USA, 2006. Association for Computing Machinery.