

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ DE TOURS

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

[polytech.univ-tours.fr](http://polytech.univ-tours.fr)

**Projet Recherche & Développement**

**2019-2020**

# **Logiciel fakenews : détecteur d'images et de vidéos contrefaites**

**POLYTECH<sup>®</sup>**  
TOURS

**Tuteur académique**  
**Donatello CONTE**

**Étudiant**  
**Alexandre BURNIER-FRAMBORET (DI5)**

1<sup>er</sup> avril 2020



## Liste des intervenants

Nom	Email	Qualité
Alexandre BURNIER-FRAMBORET	<a href="mailto:alexandre.burnier-framboret@etu.univ-tours.fr">alexandre.burnier-framboret@etu.univ-tours.fr</a>	Étudiant DI5
Donatello CONTE	<a href="mailto:donatello.conte@univ-tours.fr">donatello.conte@univ-tours.fr</a>	Tuteur académique, Département Informatique



# Avertissement

Ce document a été rédigé par Alexandre Burnier-Framboret susnommé l'auteur.

L'Ecole Polytechnique de l'Université de Tours est représentée par Donatello Conte susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



## Pour citer ce document

Alexandre Burnier-Framboret, *Logiciel fakenews : détecteur d'images et de vidéos contrefaites*,  
Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais  
de Tours, Tours, France, 2019-2020.

```
@mastersthesis{
  author={Burnier-Framboret, Alexandre},
  title={Logiciel fakenews : détecteur d'images et de vidéos contrefaites},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université de Tours},
  address={Tours, France},
  year={2019-2020}
}
```

# Table des matières

<b>Liste des intervenants</b>	<b>a</b>
<b>Avertissement</b>	<b>b</b>
<b>Pour citer ce document</b>	<b>c</b>
<b>Table des matières</b>	<b>i</b>
<b>Table des figures</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1 Contexte .....	1
2 Objectif .....	1
3 Hypothèses .....	1
4 Bases méthodologiques .....	2
<b>2 Description générale</b>	<b>3</b>
1 Environnement du projet .....	3
2 Caractéristiques des utilisateurs .....	3
3 Fonctionnalités du système .....	3
4 Structure générale du système .....	4
<b>3 État de l'art</b>	<b>7</b>
1 Périmètre.....	7
2 Réseau de neurones auto-encodeur.....	7
3 Réseau antagoniste génératif (GAN) .....	8
4 Solutions techniques des détections de "DeepFake" .....	9
5 Solutions retenues pour l'implémentation de la détection .....	13

<b>4</b>	<b>Analyse et conception</b>	<b>14</b>
1	Identification du problème .....	14
2	Le réseau de neurones convolutifs (CNN) .....	15
2.1	Conception ciblée sur la détection de l'auto-encodeur.....	16
2.2	Conception ciblée sur la détection du réseau antagoniste génératif .....	17
3	Le réseau de neurones récurrents (LSTM) .....	18
3.1	Conception ciblée sur la détection de l'auto-encodeur.....	19
4	Conception générale du système.....	20
<b>5</b>	<b>Mise en œuvre</b>	<b>22</b>
1	Pré-traitement des jeux de données.....	22
1.1	Jeux de données utilisés pour la détection de DeepFake créés par auto-encodeur.....	24
1.2	Jeux de données utilisés pour la détection de DeepFake créés par réseau antagoniste génératif (GAN).....	24
2	Métriques de la performance des modèles d'apprentissage profond.....	26
3	Implémentations du réseau de neurones ResNet50 pour la détection de DeepFake créés par auto-encodeur .....	27
4	Implémentations du réseau de neurones CNN pour la détection de DeepFake créés par GAN.....	30
5	Implémentations du réseau de neurones CNN-LSTM pour la détection de DeepFake créés par auto-encodeur.....	31
6	Protocole d'entraînement du réseau de neurones ResNet50 .....	32
7	Protocole de tests du réseau de neurones ResNet50 après entraînement .....	43
8	Protocole d'entraînement du réseau de neurones CNN .....	48
9	Protocole de tests du réseau de neurones CNN après entraînement .....	57
10	Visualisation avec l'interface.....	58
11	Analyse des résultats .....	63
12	Limites .....	63
13	Risques.....	64
14	Outils et bibliothèques.....	64
15	Choix techniques .....	64
<b>6</b>	<b>Bilan et conclusion</b>	<b>65</b>
1	Bilan du semestre 9 .....	65
1.1	Tâches terminées.....	65
1.2	Tâches qui restent à faire.....	65
2	Planning du semestre 10 .....	66
3	Bilan du semestre 10 .....	67
4	Bilan sur la qualité .....	67
5	Bilan auto-critique sur la gestion .....	67

<b>Annexes</b>	<b>68</b>
<b>A Description des interfaces externes du logiciel</b>	<b>69</b>
1 Interface matériel/logiciel.....	69
2 Interface homme/machine .....	70
3 Interfaces logiciel/logiciel .....	70
<b>B Spécifications fonctionnelles</b>	<b>71</b>
1 Vue globale des fonctionnalités du programme de détection de "DeepFake" .....	71
2 Définition de la fonction 1 : extract_content .....	71
2.1 Identification de la fonction 1 .....	71
2.2 Description de la fonction 1 .....	71
3 Définition de la fonction 2 : align_face.....	72
3.1 Identification de la fonction 2 .....	72
3.2 Description de la fonction 2 .....	72
4 Définition de la fonction 3 : blur_face .....	72
4.1 Identification de la fonction 3 .....	72
4.2 Description de la fonction 3 .....	72
5 Définition de la fonction 4 : affine_face.....	72
5.1 Identification de la fonction 4 .....	72
5.2 Description de la fonction 4 .....	72
6 Définition de la fonction 5 : extract_co_occurrence_matrix .....	73
6.1 Identification de la fonction 5 .....	73
6.2 Description de la fonction 5 .....	73
7 Définition de la fonction 6 : train_model .....	73
7.1 Identification de la fonction 6 .....	73
7.2 Description de la fonction 6 .....	73
8 Définition de la fonction 7 : analyze_content.....	74
8.1 Identification de la fonction 7 .....	74
8.2 Description de la fonction 7 .....	74
9 Définition de la fonction 8 : display_result .....	74
9.1 Identification de la fonction 8 .....	74
9.2 Description de la fonction 8 .....	74
10 Définition de la fonction 9 : display_artefact .....	74
10.1 Identification de la fonction 9 .....	74
10.2 Description de la fonction 9 .....	74
11 Définition de la fonction 10 : load_model .....	74
11.1 Identification de la fonction 10 .....	74
11.2 Description de la fonction 10 .....	75
12 Définition de la fonction 11 : save_model .....	75

12.1	Identification de la fonction 11 .....	75
12.2	Description de la fonction 11 .....	75
<b>C</b>	<b>Spécifications non fonctionnelles</b>	<b>76</b>
1	Contraintes de développement et conception .....	76
1.1	Contrainte matériels .....	76
1.2	Contrainte de langage de programmation .....	76
1.3	Contrainte logiciels .....	76
1.4	Contrainte environnementale .....	77
2	Contraintes de fonctionnement et d'exploitation .....	77
2.1	Performances .....	77
2.2	Capacités .....	77
2.3	Modes de fonctionnement .....	77
2.4	Contrôlabilité .....	77
2.5	Sécurité .....	78
2.6	Intégrité .....	78
2.7	Maintenance et évolution du système .....	78
<b>D</b>	<b>Gestion de projet</b>	<b>79</b>
<b>E</b>	<b>Cahier du développeur</b>	<b>84</b>
<b>F</b>	<b>Cahier des charges</b>	<b>90</b>
<b>G</b>	<b>Manuels d'utilisation</b>	<b>96</b>
	<b>Comptes rendus hebdomadaires</b>	<b>134</b>
	<b>Webographie</b>	<b>137</b>
	<b>Bibliographie</b>	<b>138</b>
	<b>Glossaire</b>	<b>140</b>



# Table des figures

## 2 Description générale

1	Diagramme de cas d'utilisation .....	4
2	Diagramme de composant du système .....	4
3	Diagramme de séquence du système .....	5
4	Diagramme d'activité du système .....	6

## 3 État de l'art

1	Comparaison de l'architecture d'un GAN traditionnel (a) et StyleGAN (b) avec sa génération d'images.....	8
2	Pipeline du modèle de deep learning (CNN-LSTM) .....	10
3	Graphique des résultats de la solution .....	11
4	Pipeline deep learning de la détection de "DeepFake" générés par GAN .....	12

## 4 Analyse et conception

1	Falsification du visage d'une actrice par celui de Nicolas Cage en utilisant un auto-encodeur.....	14
2	Visage créé par un réseau antagoniste génératif (styleGAN) .....	15
3	Max pooling avec un filtre $2 \times 2$ et un pas de 2.....	16
4	Processus de simulation d'un "DeepFake" auto-encodeur .....	17
5	Exemple d'une matrice de co-occurrence $0^\circ$ en niveau de gris.....	18
6	Structure d'une couche LSTM.....	19
7	Diagramme de classes du système .....	21

## 5 Mise en œuvre

1	Exemple d'une image extraite d'une vidéo falsifiée .....	22
---	--	----

2	Exemple d'un des nombreux visages extrait de l'image précédente.....	23
3	Extrait $10 \times 10$ d'une matrice $256 \times 256$ de co-occurrence de degré $0^\circ$ à partir de la matrice de couleur rouge d'une image RGB .....	23
4	Exemple d'une image venant du jeu de données DeepFake de FaceForensics++.....	25
5	Exemple d'une image créé par le logiciel StarGAN.....	25
6	Exemple d'images créés par le GAN StyleGAN2.....	25
7	Exemple d'une courbe ROC .....	26
8	Fonction d'activation sigmoïde .....	27
9	Exemple d'un aplatissement après un résultat de convolution .....	28
10	Exemples de scheduler triangulaire.....	28
11	Exemple de la différence entre un taux d'apprentissage important ou faible .....	29
12	Échantillon d'images avec de gauche à droite, UADFV, DeepFakeTIMIT HQ, DeepFakeTIMIT LQ, Facebook Challenge, DeepFake Detection, NeuralTextures ...	33
13	Courbes de perte (loss) sur la base d'entraînement et de validation.....	34
14	Courbes de précision (accuracy) sur la base d'entraînement et de validation .....	34
15	Courbe ROC sur la base d'entraînement, validation et test .....	35
16	Courbes de la perte du modèle au cours des epochs sur la base d'entraînement et validation .....	36
17	Courbes de précision (accuracy) sur la base d'entraînement et de validation .....	36
18	Courbe ROC sur la base d'entraînement, validation et test .....	37
19	Courbes de la perte du modèle au cours des epochs sur la base d'entraînement et validation .....	38
20	Courbes de précision (accuracy) sur la base d'entraînement et de validation .....	39
21	Courbe ROC sur la base d'entraînement, validation et test .....	39
22	Courbes de la perte du modèle au cours des epochs sur la base d'entraînement et validation .....	40
23	Courbes de précision (accuracy) sur la base d'entraînement et de validation .....	41
24	Courbe ROC sur la base d'entraînement, validation et test .....	41
25	Courbe ROC du modèle sur le jeu de données UADFV .....	43
26	Courbe ROC du modèle issu du papier de recherche sur le jeu de données UADFV	43
27	Courbe ROC du modèle sur le jeu de données VIDTIMIT/DeepFakeTIMIT HQ ....	44
28	Courbe ROC du modèle issu du papier de recherche sur le jeu de données DeepFakeTIMIT HQ .....	44
29	Courbe ROC du modèle sur le jeu de données DeepFakeTIMIT LQ .....	45
30	Courbe ROC du modèle issu du papier de recherche sur le jeu de données DeepFakeTIMIT LQ .....	45
31	Courbe ROC du modèle entraîné sur le jeu de données DeepFake Detection.....	46
32	Courbe ROC du modèle entraîné sur le jeu de données NeuralTextures .....	47
33	Courbe ROC du modèle entraîné sur le jeu de données NeuralTextures .....	47
34	Échantillon d'images de gauche à droite venant de styleGAN2, styleGAN, StarGAN	48

35	Courbe de la perte (loss) du modèle issu du papier de recherche sur la base d'entraînement et validation .....	49
36	Courbe de la précision (accuracy) du modèle issu du papier de recherche sur la base d'entraînement et validation .....	49
37	Courbe de perte (loss) du modèle CNN entraîné.....	50
38	Courbe de précision (accuracy) du modèle CNN entraîné.....	51
39	Courbe ROC du modèle CNN entraîné .....	51
40	Courbe de perte (loss) du modèle CNN entraîné.....	52
41	Courbe de précision (accuracy) du modèle CNN entraîné.....	53
42	Courbe ROC du modèle CNN entraîné .....	53
43	Courbe de perte (loss) du modèle CNN entraîné.....	54
44	Courbe de précision (accuracy) du modèle CNN entraîné.....	55
45	Courbe ROC du modèle CNN entraîné .....	55
46	Courbe ROC du modèle entraîné sur la base de test StarGAN .....	57
47	Courbe ROC du modèle entraîné sur la base de test StyleGAN.....	58
48	Aperçu du fonctionnement de GRAD-CAM.....	59
49	Échelle de couleurs.....	59
50	Zones d'intérêt dans la détection d'un DeepFake créé par auto-encodeur.....	60
51	Zones d'intérêt de la détection d'un visage non falsifié par le détecteur de Deep-Fake créés par auto-encodeur.....	60
52	Zones d'intérêt de la détection d'un visage non falsifié par le détecteur de Deep-Fake créés par GAN .....	61
53	Zones d'intérêt dans la détection d'un DeepFake créé par GAN .....	61
54	Interface utilisateur .....	62
 <b>6 Bilan et conclusion</b>		
1	Aperçu des tâches existantes grâce à un "Trello" .....	66
 <b>A Description des interfaces externes du logiciel</b>		
1	Comparaison dans le temps (entre 2003 et 2013) de la bande passante entre CPU et GPU.....	69
2	Maquette de l'interface homme/machine.....	70
 <b>B Spécifications fonctionnelles</b>		
1	Arbre hiérarchique des fonctionnalités .....	71
2	Formule du filtre appliqué sur chaque pixel .....	72
3	Utilisation des points du contours visage pour aligner l'image sur l'autre .....	73
4	Formule de la matrice de co-occurrence C .....	73

**E Cahier du développeur**

1	Diagramme de classes du système .....	84
---	---------------------------------------	----

# 1

## Introduction

Ce sujet de recherche et développement est proposé par Monsieur Donattelo Conte, qui a pour but l'identification d'images et vidéos contrefaites aussi appelées "DeepFake".

### 1 Contexte

Apparu initialement de manière fictive en 2008, le phénomène du "DeepFake" s'est accéléré à partir de 2016, avec l'apparition de logiciels libre permettant sa réalisation sans connaissance particulière. Le "DeepFake" (mot valise entre deep learning et fake) consiste généralement à partir d'une image ou vidéo à substituer par des méthodes d'apprentissage profond le visage de la personne-cible par celui d'un autre individu existant ou créée artificiellement. Ce phénomène devenant alors particulièrement inquiétant au niveau sociétal par ses possibles effets, de nombreux acteurs privés et scientifiques travaillent activement à détecter de manière fiable ce type de contenu.

### 2 Objectif

Ce projet vise à fournir un détecteur de "DeepFake", se focalisant en particulier sur les images et vidéos contenant des visages d'individu qui ont été falsifiés par un réseau de neurones de type **auto-encodeur** ou **réseau antagoniste génératif** (GAN).

Les objectifs peuvent se résumer ainsi :

- Détecter la falsification de l'image d'un individu
- Détecter la falsification vidéo d'un individu
- Afficher les artefacts de l'image qui ont permis de conclure sur la potentielle falsification

### 3 Hypothèses

L'hypothèse est que les méthodes d'apprentissage profond qui seront implémentées et issues des documents de recherche sur la détection de "DeepFake" seront fonctionnelles lors de leur utilisation. Si lors de leur implémentation, le résultat attendu ne convient pas à un certain pourcentage de validation (<90%) lors des tests alors d'autres solutions non considérées pour ce

projet seront étudiés.

En fonction du temps, il est possible qu'un choix de priorité soit opéré dans l'implémentation des algorithmes pour la détection d'images et que donc certains ne puissent finalement pas être implémentés.

## 4 Bases méthodologiques

La gestion du projet se fera en méthode Agile. Ainsi, un ensemble de jalons lors du développement sera créé ainsi que la mise en place de rapports hebdomadaires destinés au tuteur pédagogique afin de suivre de près l'avancement du projet avec des comptes-rendus de chaque réunion avec celui-ci. Un diagramme de Gantt prévisionnel sera utilisé afin de situer et répartir la charge de travail dans le temps. Enfin, un outil de type « Trello » sera utilisé afin de découper le projet en tâches à réaliser.

La norme de programmation de Python sera PEP-8. La modélisation des fonctionnalités du système se fera par différents diagrammes UML créés par Astah. Le code source du logiciel sera hébergé sur un répertoire personnel public GitLab et la gestion des versions par Git.

# 2

## Description générale

### 1 Environnement du projet

Aucun matériel ou logiciel préexistant n'est à signaler pour ce projet. De plus, il n'a pas objectif à s'inscrire dans un quelconque environnement déjà existant. Néanmoins, les tests des algorithmes de détection sur les contenus générés par **auto-encodeur** et GAN seront effectués sur des "DeepFake" créés par le logiciel FaceSwap et StarFace dans un premier temps avant d'être possiblement effectués sur d'autres jeux de données.

### 2 Caractéristiques des utilisateurs

Le principal utilisateur est un utilisateur potentiel qui est représenté par les enseignants-chercheurs de l'école de journalisme de Tours. Ce sont des utilisateurs professionnels qui ont une connaissance basique dans l'informatique. Pour ces utilisateurs, une interface graphique pour l'utilisation du logiciel sera privilégiée. Il sera néanmoins possible d'utiliser le logiciel en mode ligne de commande. Le logiciel sera uniquement accessible sur les machines de ces utilisateurs et ne bénéficie pas d'une protection particulière (ex : login, mot de passe).

### 3 Fonctionnalités du système

Le diagramme des cas d'utilisations ci-dessous décrit l'usage du système par l'utilisateur potentiel. Ainsi pour résoudre le problème de connaître si le contenu est falsifié ou non, l'utilisateur devra dans un premier temps indiquer son type d'entrée (vidéo ou image) qui sera ensuite analysé par le logiciel et enfin un résultat sera affiché sous la forme d'un indicateur avec si possible l'indication des artefacts qui ont permis aux **réseaux de neurones** de conclure.

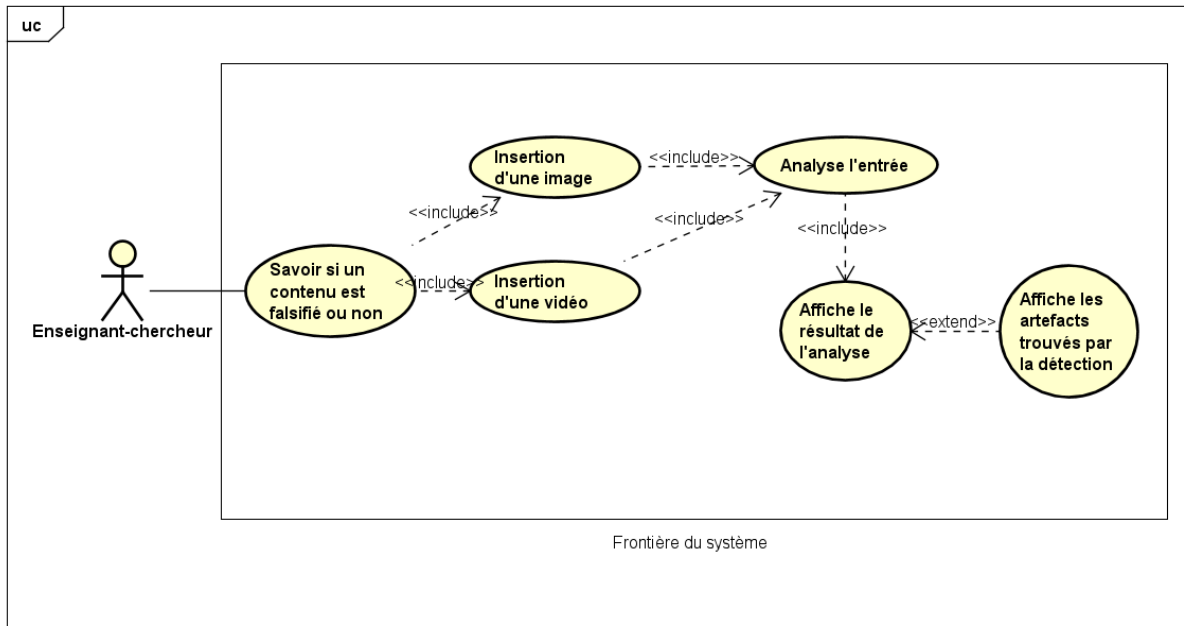


Figure 1 – Diagramme de cas d'utilisation

## 4 Structure générale du système

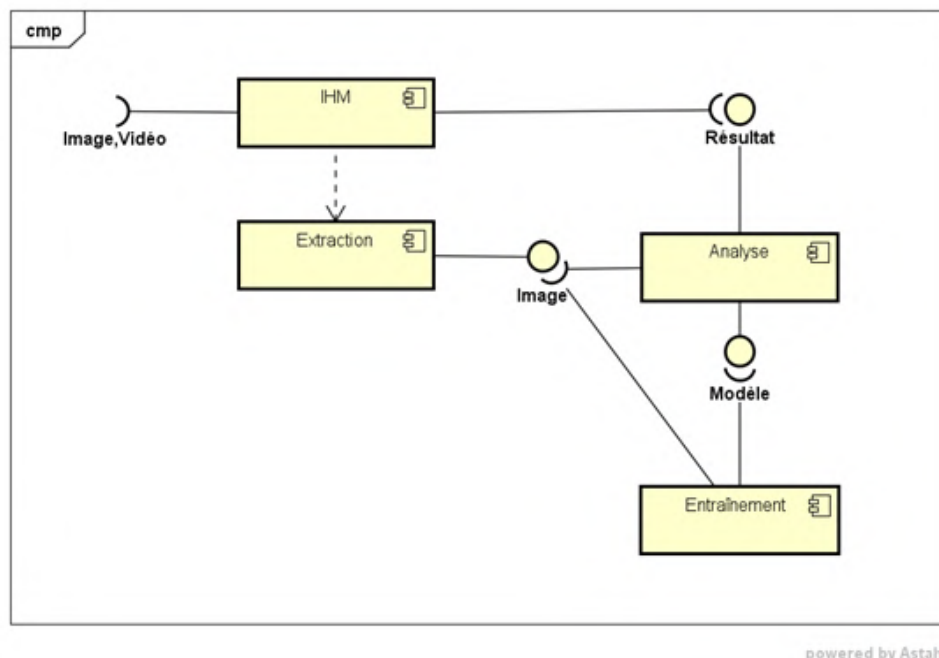


Figure 2 – Diagramme de composant du système

Le diagramme composant ci-dessus permet de comprendre dans la globalité, le fonctionnement interne du logiciel. Ainsi, l'IHM permettra à l'utilisateur d'intégrer ses entrées dans le système, l'extraction permettra ensuite d'extraire la zone d'intérêt de l'image, de découper en plusieurs images si sous une forme vidéo et de pouvoir faire un alignement afin d'obtenir une donnée



exploitable pour le modèle d'apprentissage profond. Par la suite, ces données seront utilisées par l'analyse pour déterminer si le contenu est falsifié. Le composant Entraînement ne sera utilisé que dans le cadre du développement afin que l'utilisateur puisse utiliser un modèle de deep learning pré-entraîné dont il n'aura pas à se préoccuper.

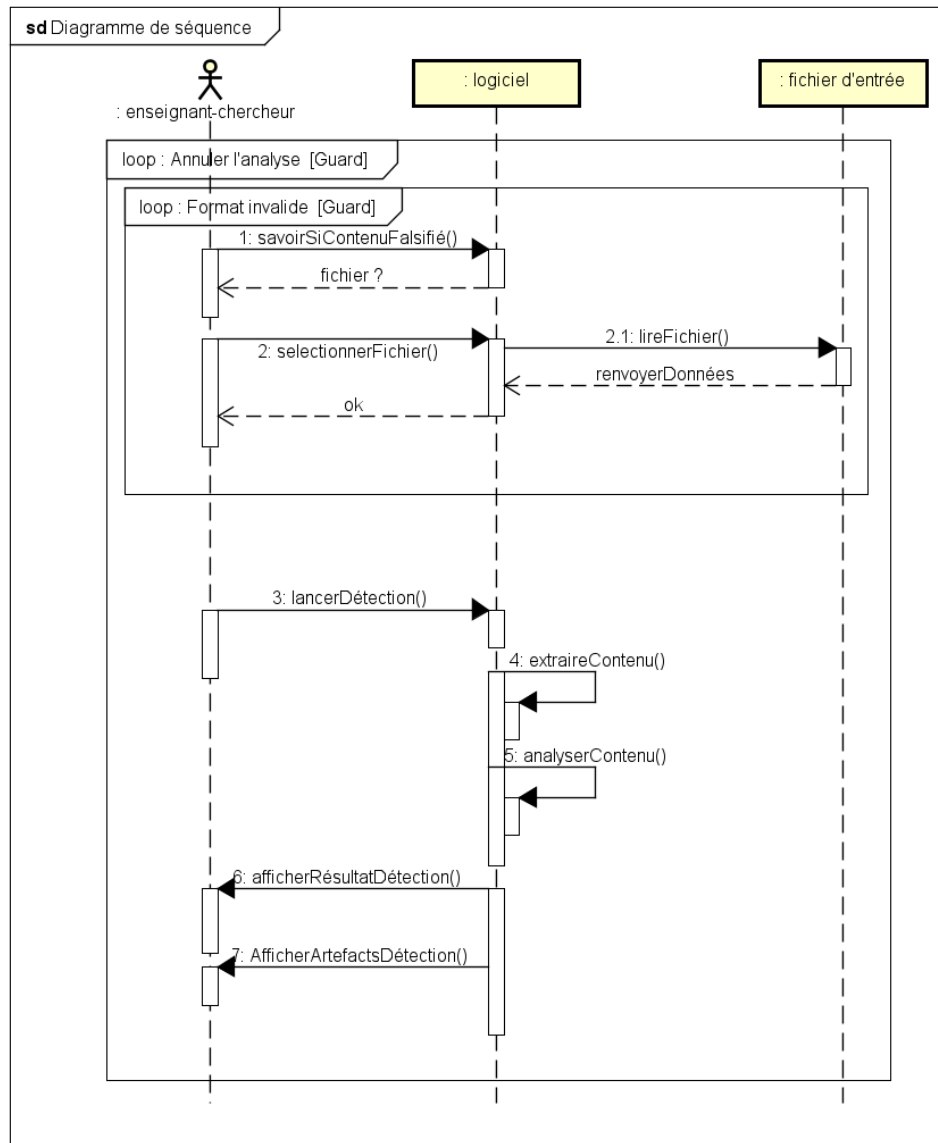


Figure 3 – Diagramme de séquence du système

Le diagramme de séquence ci-dessus permet de décrire les étapes ainsi que les différentes interactions entre le logiciel et l'environnement extérieure. Ainsi, l'utilisateur potentiel qui souhaite savoir si le contenu de son entrée est falsifié devra au préalable renseigner un format reconnu par le logiciel sous peine de devoir recommencer le processus. L'utilisateur pourra ensuite déclencher la détection qui sera suivi par l'extraction du contenu de l'entrée, de son analyse et enfin de l'affichage du résultat sous forme d'indicateurs visuels et numériques.

Le diagramme d'activité ci-dessous permet de décrire de façon détaillé chaque action du programme en fonction de la situation lors de son fonctionnement. Ainsi, l'utilisateur pourra dans certains cas annuler le processus de détection du programme au moyen d'un bouton (lors de l'extraction ou analyse du contenu).

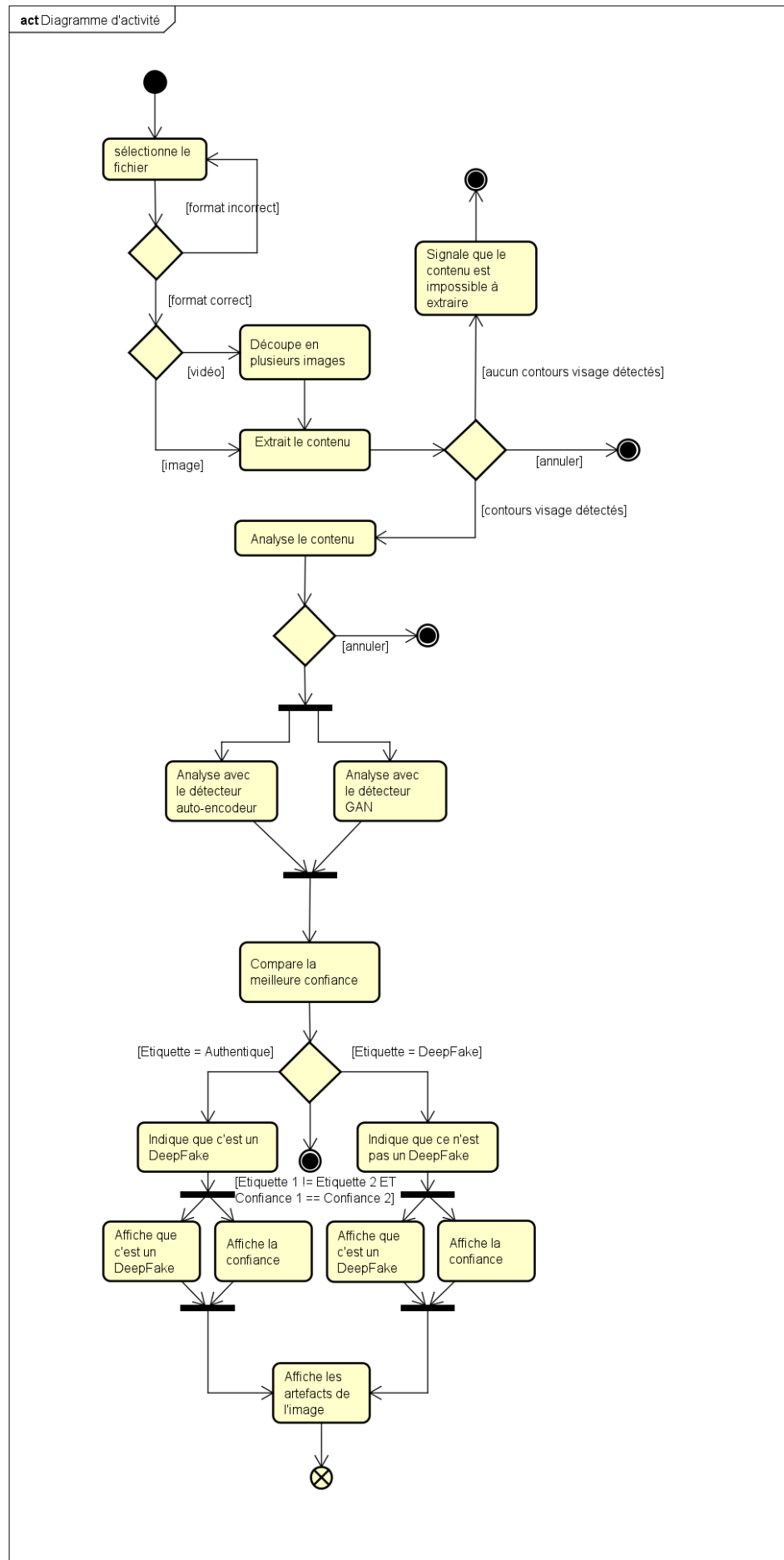


Figure 4 – Diagramme d'activité du système

# 3

## État de l'art

### 1 Périmètre

Dans ce chapitre, l'intérêt sera uniquement porté sur les méthodes de détection de "DeepFake" qui ciblent spécifiquement les individus à l'aide du deep learning dans le but de remplacer/modifier le visage de la personne-cible par celui d'un autre. Afin de délimiter le sujet, la discussion portera uniquement sur les types de réseaux de neurones qui sont le plus communément utilisés, à savoir **auto-encodeur** et le réseau antagoniste génératif (GAN) qui permet la génération d'images possédant un fort réalisme.

### 2 Réseau de neurones auto-encodeur

L'auto-encodeur est un type de réseau de neurones simple à trois couches [WWW1]. Il existe aussi plusieurs autres implémentations tel que l'auto-encodeur avec débruitage et l'auto-encodeur profond supervisé. Le but principal de l'auto-encodeur est de créer une nouvelle représentation d'un jeu de données. Il se présente sous la forme d'un encodeur et décodeur. L'encodeur permet à partir d'un jeu de données, de réduire sa dimension (supprimer un certain nombre de descripteurs). Le décodeur permet de reconstruire la donnée à partir de la nouvelle représentation générée par l'encodeur tout en essayant de minimiser l'erreur.

Dans le cas de la génération de "DeepFake", l'astuce est alors que l'encodeur soit commun aux deux images de visage A et B (le décodeur est indépendant pour chaque visage), ce qui lui permet d'apprendre (observer et décrire) les caractéristiques de chacun (sourire, luminosité, regard, ...) en ayant une "description mutuel commune". Le décodeur du visage A peut alors le recréer sur B en respectant les caractéristiques de celui-ci, l'encodeur fournissant une description commune grâce au procédé décrit au-dessus. Ce processus est équivalent pour le décodeur du visage B.

Au niveau mathématique, cela se décrit tel que :

L'encodeur calcule le nouveau vecteur  $h$  à partir de  $x$  où  $W$  est une matrice de taille  $m \times n$  et  $b$  un vecteur de biais de taille  $m$ .  $\sigma$  est la fonction d'activation de type tangente hyperbolique définie par :

$$\sigma(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}}$$

où

$$h = \sigma(Wx + b) \quad (1)$$

Le décodeur cherche à reconstruire le vecteur  $x$  à partir de  $h$  où  $\tilde{x}$  est un vecteur de taille  $m$ ,  $W$  une matrice de poids de taille  $n \times m$  et  $b$  un vecteur de biais de taille  $n$ .

$$\tilde{x} = \sigma(Wh + b) \quad (2)$$

Pour réduire l'erreur de reconstruction de  $l$ , l'erreur quadratique moyenne est utilisé entre  $x$  et  $\tilde{x}$  avec les paramètres  $\theta = W, b, W, b$ .

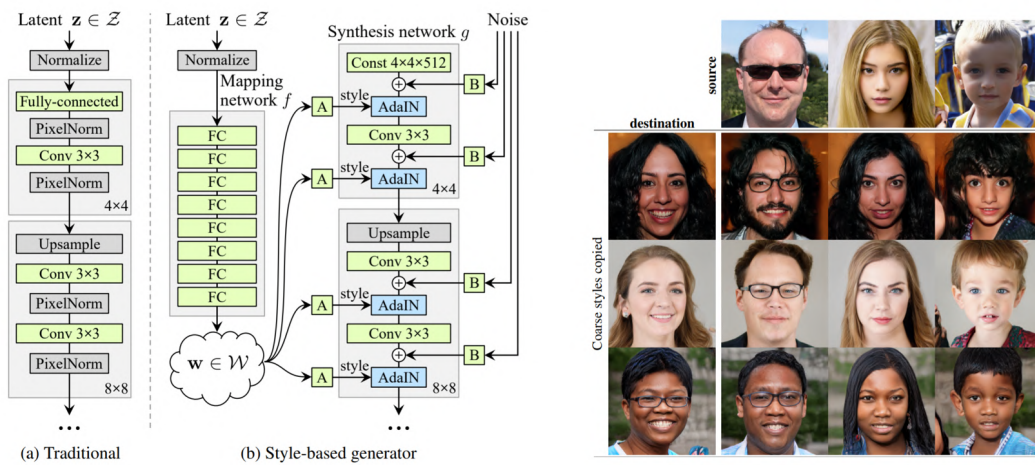
$$L_{\text{MSE}}(\theta) = \frac{1}{d} \sum_{x \in D} \|x - \tilde{x}\|^2 \quad (3)$$

### 3 Réseau antagoniste génératif (GAN)

Le réseau antagoniste génératif est un type de réseau de neurones à l'apprentissage non supervisé. Découverte relativement récente (2014), il permet principalement la génération d'images avec un fort degré de réalisme grâce à l'application d'un scénario qui consiste en un entraînement réciproque par paires de réseaux de neurones (appelée Générateur et Discriminateur). Le Générateur génère les images que reçoit le Discriminateur et qu'il compare à une base de données d'images réels dans le but d'avoir la capacité de reconnaître le réel du faux. A la suite de cela, une boucle de feedback permet au Générateur de connaître les images sur lesquelles le Discriminateur n'a pas été "trompé", dans le même temps le Discriminateur reçoit ceux pour lesquels sa classification s'est avérée fautive, permettant à ces deux réseaux de neurones de s'entraider dans l'amélioration de leurs générations ou identifications. Ce processus est issu de la théorie des jeux [6]. Dans le cas de la génération d'un visage, il prend en entrée un certain nombre de **features** désirés à appliquer sur la création (couleur des cheveux, couleur des yeux, âge, ...) et avec le visage de base, lui applique diverses transformations en respectant les features en entrée.

Formule mathématique du scénario de la théorie des jeux utilisé :

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$



**Figure 1** – Comparaison de l'architecture d'un GAN traditionnel (a) et StyleGAN (b) avec sa génération d'images

## 4 Solutions techniques des détections de "DeepFake"

L'utilisation des modèles de deep learning, nécessite d'utiliser des jeux de données à des fins d'entraînement (avec un possible pré-traitement dans la création des jeux de données "DeepFake"), de validation et test. Les jeux de données utilisés dans l'ensemble des études sont :

- CEW (2423 images de visages réels)
- EEG Eye State (14980 images de visages réels)
- DARPA GAN Challenge (241 images réels et 252 images "DeepFake" d'individus)
- UADFV (98 vidéos dont 49 non falsifiées et 49 "DeepFake" d'individus)
- VidTIMIT (44 fichiers vidéo avec audio de 43 personnes)
- DeepfakeTIMIT (320 vidéos "DeepFake" d'individus)
- AMI (100 heures vidéo de multiples réunions de personnes)
- GRID (16200 fichiers contenant des vidéos d'individus (hommes, femmes) avec l'audio)
- CelebA (202599 images de visages de célébrités)
- cycleGAN (plus de 10000 images de contenu divers tel que des objets, animaux, paysages, ...)
- HOHA (240 séquences courtes issus de 32 films de divers genres)

Certaines des solutions ont en commun des jeux de données dont il peut être intéressant de mettre en confrontation par leur implémentation.

### Solution 1

Les algorithmes de "DeepFake" avec auto-encodeur ne peuvent que synthétiser des images d'une taille fixe et doivent utiliser une déformation affine des contours pour que cette image se fonde dans le reste du visage original. Ces déformations affines sont des artefacts visibles à cause de l'inconsistance entre la résolution du faux visage et l'original. Le constat est donc de révéler ce phénomène à l'aide d'un modèle CNN permettant de décomposer en plusieurs couches l'image et de reconnaître les déformations du contour. La particularité de cette méthode est de se passer des "logiciels de DeepFake" pour créer un jeu de données d'entraînement. La simplification du processus se fait en simulant les inconsistances des déformations affines du visage, par la détection de celui-ci et l'application d'un filtre gaussien qui simule l'approximation de l'auto-encodeur ensuite aligné et ré-appliqué sur le visage de l'image originale en utilisant l'inverse de la matrice de transformation estimée avec pour résultat les mêmes inconsistances que pour une image "DeepFake" créée par auto-encodeur. Afin d'élargir le jeu de données, de multiples changements sont réalisés dans les images au niveau du contraste, de la luminosité, distorsion, finesse, couleurs et forme du lissage affine [17].

Tableau des performances des modèles CNN :

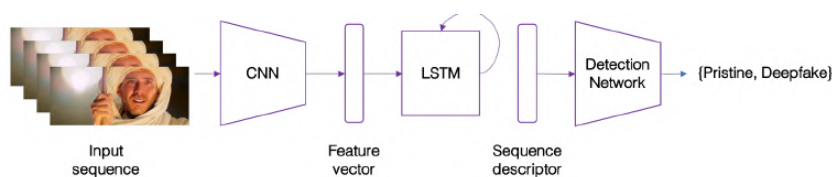
Méthodes	UADFV	LQ	HQ
Two-stream NN	85.1	83.5	73.5
Meso-4	84.3	87.8	68.4
MesoInception-4	82.1	80.4	62.7
Ours-VGG16	84.5	84.6	5.4
Ours-ResNet50	<b>97.4</b>	<b>99.9</b>	<b>93.2</b>
Ours-ResNet101	95.4	97.6	86.9
Ours-ResNet152	93.8	99.4	91.2

Le test est réalisé sur UADFV et DeepFakeTIMIT où il est possible de constater que le modèle CNN ResNet50 est le plus performant à la détection sur l'ensemble des jeux de données. Ce modèle a été entraîné sur 20 **epoch**.

### Solution 2

Dans cette étude qui se focalise sur l'auto-encodeur, le modèle deep learning est sur deux niveaux d'analyses ; l'utilisation d'un CNN afin d'extraire les features au niveau de l'image suivi par un LSTM (2048 dimensions de features) qui permet d'avoir "une mémoire dans le temps" suivi par 512 neurones complètement connectées, qui capturent les inconsistances temporelle introduit par le changement de visage et permet de connaître si l'image est issue d'une falsification par auto-encodeur grâce à la couche softmax en sortie.

Le protocole de l'étude est 70/15/15 soit 70% entraînement, 15% test et 15% validation sur le modèle. Le jeu de données utilisé est composé de 300 vidéos "DeepFake" trouvés sur internet et 300 vidéos venant de HOHA qui contient uniquement des séquences vidéos non falsifiées de personnes [4].



**Figure 2** – Pipeline du modèle de deep learning (CNN-LSTM)

Tableau des performances du modèle CNN-LSTM :

Modèle	Entraînement %	Validation %	Test %
CNN-LSTM (20 images)	99.5	96.9	96.7
CNN-LSTM (40 images)	99.3	97.1	97.1
CNN-LSTM (80 images)	99.7	97.2	97.1

Selon l'étude, le modèle CNN-LSTM permet d'obtenir une détection de "DeepFake" d'au moins 96.7% (10 epochs) avec seulement 20 frames, ce qui le rend particulièrement pratique si la donnée en entrée est pauvre ( $\pm 2$  secondes de vidéo).

### Solution 3

Une autre méthode consiste à détecter l'incohérence entre l'audio et la vidéo notamment au niveau de la synchronisation labiale et du doublage. Les datasets utilisés sont issus de véritables vidéos, qui ont ensuite été altérés pour démontrer la possibilité de détection par l'utilisation d'OpenPose et face landmarks avec la mesure du delta de 42 distances différentes entre 48 à 64 points remarquable. Il est à noter que l'étude porte uniquement sur des visages de face dû à la contrainte liée aux mesures. Les features visuels et audio sont ensuite concaténés dans un joint-vecteur. La PCA est utilisée pour réduire le nombre de features. Pour l'audio, MFCC permet d'extraire les features audio. Ensuite, le modèle est entraîné avec les données du joint-vecteur. De nombreux types de réseaux de neurones sont possibles comme LSTM, SVM, CNN, GMM. L'étude montre que le LSTM est celui qui performe le mieux sur l'ensemble des jeux de données testés (VidTIMIT, AMI, GRID) [10].

Tableau résumé des résultats sur différents modèles :

Modèle	Entraînement (% erreur)	Test (% erreur)
LSTM(64), blk5, pca60, win10	0.56	24.74
LSTM(32), blk5, pca60, win10	1.26	24.74
MLP (64), blk5, pca60, win10	9.03	53.45
SVM, blk2, pca60	27.36	56.18
SVM, blk5, pca60	3.65	30.36
SVM, blk5, pca100	0.00	45.28
GMM32, blk5, pca60	2.62	56.09

Les résultats des tests sur les différents jeux de données n'ont pas un pourcentage supérieur à 56.18 % qui comparés à d'autres méthodes se concentrant uniquement sur l'image sont bien plus performantes.

#### Solution 4

Une étude démontre que le clignement des yeux peut être utilisé afin de détecter une falsification "DeepFake". Cela repose sur l'utilisation d'un LSTM sur un ensemble de données d'entraînement correspondant à une région découpée de l'image soit la zone de l'œil, avec 0 ou 1 en fonction de son état ouvert ou non. En entrée, un CNN est utilisé avec (backpropagation) rétropropagation du gradient avec un LSTM-RNN en sortie. Plusieurs techniques sont utilisées comme la probabilité du clignement de l'œil par rapport à une situation normale ou la consistance du ratio et de l'emplacement des yeux [16].

Néanmoins l'article [9] présente l'étude avec une démarche non concluante sur un réseau de neurones de type GAN. Cette méthode est donc encore sujette à être prouvée sur ce sujet. Par ailleurs, l'étude conclut simplement sur le fait que le LSTM échoue à détecter les incohérences entre le mouvement labiale et la parole, le SVM ne peut détecter que 8.97 % d'un "DeepFake" en haute définition. Les tests ont été fait sur VidTIMIT avec une génération de "DeepFake" pour l'entraînement grâce à deux modèles GAN différents.

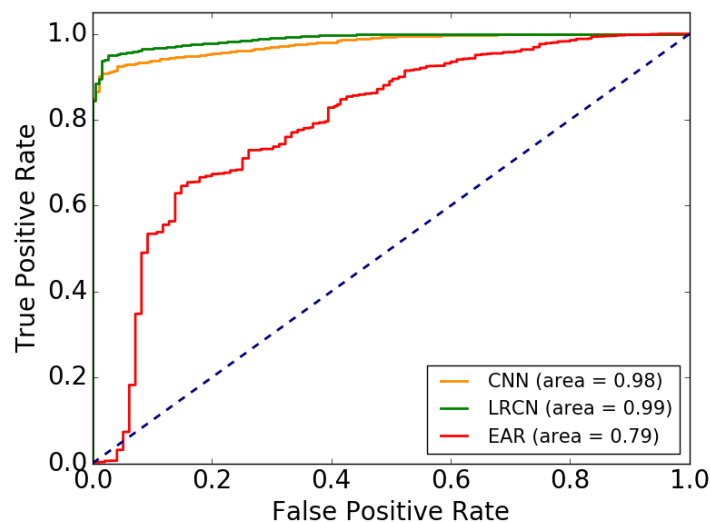


Figure 3 – Graphique des résultats de la solution

**Solution 5**

Cette méthode consiste à déceler les "DeepFake" par la détection des inconsistances dans le déplacement de la tête (en 3D). Pour cela, un SVM est entraîné selon plusieurs critères tel que la distance du cosinus, rotation, translation entre deux visages et permet à la fin d'obtenir une différence dans la distribution des distances cosinus entre une vidéo avec un visage réel et un "DeepFake" [15].

Tableau de l'évaluation AUROC :

features	frame	video
$\vec{v}_a - \vec{v}_c$	0.738	0.888
$\vec{r}_a - \vec{r}_c$	0.798	0.898
$\vec{R}_a - \vec{R}_c$	0.853	0.913
$(\vec{v}_a - \vec{v}_c) \& (\vec{t}_a - \vec{t}_c)$	0.840	0.949
$(\vec{r}_a - \vec{r}_c) \& (\vec{t}_a - \vec{t}_c)$	0.866	0.954
$(\vec{R}_a - \vec{R}_c) \& (\vec{t}_a - \vec{t}_c)$	0.890	0.974

Selon l'évaluation AUROC de l'étude, le SVM permet une détection de 89% des "DeepFake" créés par auto-encodeur et 84.3% sur des "DeepFake" GAN (le floutage diminue l'estimation de la rotation et de la translation de la tête). Les tests ont été réalisés sur VidTIMIT et DARPA GAN Challenge.

**Solution 6**

Cette dernière méthode [11] consiste à déceler les "DeepFake" générés par GAN grâce à l'utilisation de la matrice de co-occurrence RGB de l'image dans l'entraînement d'un CNN. Le protocole d'apprentissage est le suivant :

1. Calcul des matrices de co-occurrence de chaque pixel de l'image RGB (3×256×256)
2. Envoie des matrices calculées dans le réseau de neurones convolutifs

Le pipeline du modèle CNN contient 32 neurones 3×3 convs + une couche ReLu + 32 neurones 5×5 convs + une couche de max pooling + 64 neurones 3×3 convs + une couche ReLu + 64 neurones 5×5 convs + une couche de max pooling + 128 neurones 3×3 convs + une couche ReLu + 128 neurones 5×5 convs + une couche de max pooling + 256 neurones dense + 256 neurones dense + une couche sigmoïde. L'optimiseur est une variante du gradient stochastique. Cette méthode peut traiter plus que la création artificielle de visages parmi les autres catégories d'images tel que les objets, paysages, animaux... Néanmoins cela ne sera pas abordé dans le cadre du projet de recherche.

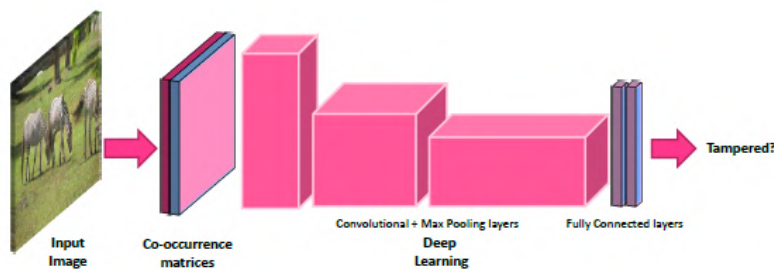


Figure 4 – Pipeline deep learning de la détection de "DeepFake" générés par GAN



Tableau des résultats :

Jeux de données	Test	Précision de détection (en %)
cycleGAN	StarGAN	99.49
StarGAN	cycleGAN	93.42

L'étude démontre que le modèle CNN utilisé permet une détection avec la matrice de co-occurrence de 99.49% concernant les visages d'individus et de 93.49% sur le reste (objets, animaux, paysages, ...) avec 50% en entraînement, 25% en validation et 25% en test. Le modèle a été entraîné à 50 epochs, sur un jeu de données pour les visages de 19 990 images, dont 17 991 générées grâce au logiciel StarGAN (variation par la couleur des cheveux, de la peau, de l'âge et du sexe). Le modèle a aussi été entraîné sur 36 302 images venant de cycleGAN (18 151 GAN et 18 151 non-GAN) contenant des objets, animaux, paysages... Enfin, il est intéressant de noter qu'un autre test de performance a été réalisé avec le même protocole mais en utilisant uniquement des images avec une compression JPEG. Malgré un facteur de qualité de seulement 75%, la détection est de 87.31%, ce qui démontre la robustesse d'une métrique basée sur la matrice de co-occurrence. Il est néanmoins démontré que si les images en entrée sont compressées, cela nécessite de faire un entraînement avec le même type de contenu pour obtenir un taux de détection satisfaisant (>90%).

Tableau des résultats avec une compression JPEG (en %) :

Facteur de qualité JPEG	Test (image originale)	Test (image compressée)
95	74.5	93.78
85	69.46	91.61
75	64.46	87.31

## 5 Solutions retenues pour l'implémentation de la détection

Afin de respecter une qualité dans la détection d'au moins 90% en précision, l'implémentation sera faite avec la solution 1 et 2 pour la détection des images générées par l'auto-encodeur qui permet par la même de les mettre en concurrence sur leur performance. La solution 6 sera implémentée pour la détection d'images générées par le réseau antagoniste génératif. Ce choix a pu être fait par rapport aux pourcentage (>90%) dans les tests ainsi que au temps estimé de la réalisation du modèle de la solution.

# 4

## Analyse et conception

Dans ce chapitre, seront abordées l'analyse et la conception au niveau technique et mathématique des implémentations retenues lors de l'état de l'art sur la détection des différents types de "DeepFake".

### 1 Identification du problème

Le problème à résoudre est le suivant :

- Détecter le "DeepFake" d'images ciblant le visage d'un individu (auto-encodeur ou GAN)
- Détecter le "DeepFake" de vidéos ciblant le visage d'un individu (auto-encodeur ou GAN)
- Montrer les artefacts de l'image qui ont permis au modèle de conclure



**Figure 1** – Falsification du visage d'une actrice par celui de Nicolas Cage en utilisant un auto-encodeur



Figure 2 – Visage créé par un réseau antagoniste génératif (styleGAN)

Afin de résoudre ce problème, les solutions retenues lors de l'état de l'art seront implémentées, à savoir un réseau de neurones convolutifs (CNN) pour la détection des images créées par l'auto-encodeur et un autre pour la détection des images créées par un réseau antagoniste génératif. Une deuxième implémentation de la détection d'images créées par l'auto-encodeur sera réalisée à l'aide d'un réseau de neurones récurrents convolutionnels (CNN-LSTM).

## 2 Le réseau de neurones convolutifs (CNN)

Le réseau de neurones convolutifs est un réseau de neurones artificiels acycliques (feed-forward) pour lequel le motif neurone est inspiré du cortex visuel des animaux. Ce type de réseau de neurones consiste en un empilage multicouche de perceptrons. Son application est principalement dans la reconnaissance d'images et vidéos, les systèmes de recommandation et le traitement du langage naturel. Un modèle CNN est composé des couches suivantes :

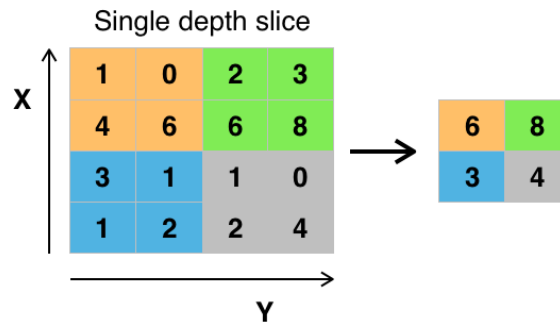
1. couche de convolution (traite l'entrée)
2. couche de pooling (compresse l'information de l'image)
3. couche de correction (fonction d'activation (Unité de rectification linéaire))
4. couche « entièrement connectée »
5. couche de perte

**la couche de convolution** possède plusieurs hyperparamètres (paramètres de distribution) nécessaire à son utilisation :

1. profondeur de la couche (nombre de neurones associés à un même champ récepteur)
2. pas (contrôle le chevauchement des champs récepteurs)
3. marge (contrôle la dimension spatiale du volume de sortie)

Les paramètres de filtrage entre les neurones d'une même couche sont partagés afin de pouvoir appliquer le même patch sur toute l'image.

**la couche de pooling** est un sous-échantillonnage d'image qui permet de réduire la quantité de paramètres et calculs du réseau de neurones. L'utilisation est en général le max pooling (valeur maximum en entrée) mais d'autres existent tel que average pooling (moyenne des valeurs de la feature map) ou stochastic pooling (valeurs aléatoires de la feature map). La forme du filtre est différente en fonction du contenu de l'image à traiter (chiffre, paysage, ...) et plus la forme sera grande, plus la perte d'information deviendra importante. Par ailleurs, le nombre de filtres doit être en fonction du produit du nombre de pixels et le nombre de caractéristiques de l'image afin d'égaliser le calcul à travers les couches car la taille des images intermédiaire diminue avec la profondeur du traitement.

Figure 3 – Max pooling avec un filtre  $2 \times 2$  et un pas de 2

**la couche de correction** permet d'améliorer l'efficacité du traitement du réseau de neurones convolutifs par une fonction d'activation (généralement non linéaire) à la sortie qui peuvent être :

- correction ReLU  $f(x) = \max(0, x)$
- correction par tangente hyperbolique  $f(x) = \tanh x$
- correction par tangente hyperbolique saturante  $f(x) = |\tanh x|$
- correction par fonction sigmoïde  $f(x) = \frac{1}{1+e^{-x}}$

**la couche de perte** est utilisée pour pénaliser l'écart entre le signal prévu et réel. C'est la dernière couche du réseau qui permet d'obtenir le résultat de l'entraînement en sortie. Ainsi, il existe différentes pertes en fonction des besoins de sortie tel que :

- perte Softmax (prédire une classe parmi K classes mutuellement exclusives)
- perte par entropie croisée sigmoïde (prédire K probabilités indépendantes dans  $[0,1]$ )
- perte euclidienne (valeurs réelles dans  $[-\infty, +\infty]$ )

## 2.1 Conception ciblée sur la détection de l'auto-encodeur

Afin d'utiliser le modèle à neurones convolutifs, il est nécessaire de l'entraîner avec les données sur lesquelles il devra s'appuyer ensuite pour déterminer si le contenu en entrée est positif ou négatif. Il est donc nécessaire de créer une base d'exemples positifs et négatifs, or la génération d'un nombre conséquent de négatifs ( $\pm 20000$  images) requiert l'utilisation des logiciels "DeepFake". La conversion d'une image en "DeepFake" prend un temps assez important qui peut être évité par la simple simulation rapide d'un auto-encodeur. Pour cela, un pipeline qu'il sera éventuellement possible d'utiliser sera créé en entrée avec les étapes suivantes :

1. extraction du visage présent dans le contenu à de multiples échelles
2. floutage du visage avec filtre ( $5 \times 5$ )
3. application du visage flouté sur le visage du contenu original
4. application de multiples modifications sur le contraste, les couleurs, la forme de la zone floutée, etc...

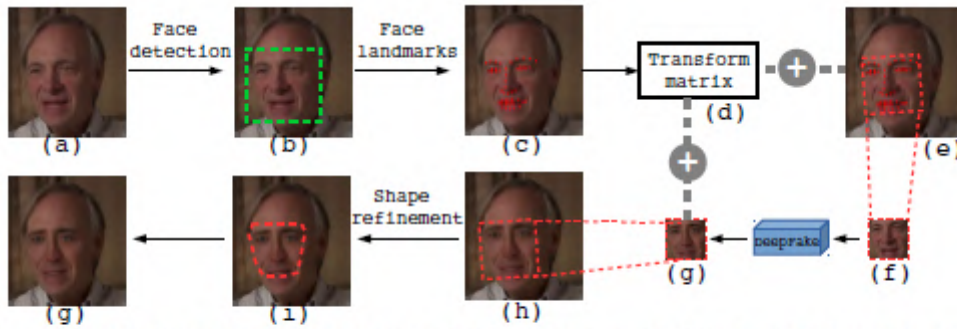


Figure 1. Overview of the DeepFake production pipeline. (a) An image of the source. (b) Green box is the detected face area. (c) Red points are face landmarks. (d) Transform matrix is computed to warp face area in (e) to the normalized region (f). (g) Synthesized face image from the neural network. (h) Synthesized face warped back using the same transform matrix. (i) Post-processing including boundary smoothing applied to the composite image. (j) The final synthesized image.

Figure 4 – Processus de simulation d'un "DeepFake" auto-encodeur

Après une génération conséquente d'images négatives avec l'utilisation de ce pipeline ou d'un logiciel de "DeepFake", l'entraînement peut commencer pour le modèle. Au vue de la quantité de données, le temps d'entraînement pourra être assez long et donc nécessiter possiblement l'utilisation d'une machine avec un GPU possédant un nombre important de FLOPS (nombre d'opérations en virgule flottante par seconde).

Le modèle de deep learning s'appuiera sur le modèle pré-existant ResNet50 (poids ImageNet) avec un entraînement sur 20 epochs. Enfin, il sera si possible affiné avec une stratégie de "hard mining" [1] (consiste à mal étiqueter les exemples positifs avec une détection de faux supérieur à 0.6 et exemples négatifs avec une détection de faux inférieur à 0.5) combinée à un taux d'apprentissage de 0.0001 sur un entraînement de 20 epochs. Après la phase d'entraînement et la validation des tests, le modèle est prêt à être utilisé par l'utilisateur avec ses propres entrées (image ou vidéo).

## 2.2 Conception ciblée sur la détection du réseau antagoniste génératif

Lors de la création d'un visage de synthèse à partir de différentes features, l'image créée par le GAN ne possède pas les mêmes statistiques des pixels comparées à celles d'une image normale. Une méthode efficace pour détecter ce genre de manipulation vient du champ de la stéganographie qui utilise la matrice de co-occurrence (généralement en niveau de gris) de l'image afin de déterminer si un message est potentiellement caché. Dans le cas actuel, cela se fera en appliquant à l'image le calcul de la matrice de co-occurrence sur les trois canaux (R,G,B).

Le calcul de la matrice s'appuie sur le respect d'une relation géométrique simple entre deux pixels  $(x_1, y_1)$   $(x_2, y_2)$ . Pour chaque couleur, il existe 4 matrices ( $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ ,  $135^\circ$ ) possibles qui résulte de la façon dont le calcul est effectué (distance entre deux pixels horizontaux, verticaux, en diagonale, ...).

Une fois les matrices de co-occurrence de l'image calculées, celles-ci sont passées dans un CNN qui va apprendre les features importantes (contraste, corrélation, entropie, ...) des textures de l'image (image positive ou négative) afin de pouvoir ensuite être capable de déterminer par la matrice de co-occurrence de l'image en entrée si celle-ci est issu d'un GAN. Les différentes formules de calcul de la matrice de co-occurrence pour une couleur :

Soit  $C_R(i, j)$  la matrice de co-occurrence initialisée à 0, le remplissage de la matrice se fait selon la formule mathématique ci-dessous :

$$C_R(f(x_1, y_1), f(x_2, y_2)) = C_R(f(x_1, y_1), f(x_2, y_2)) + 1$$

La formule d'une des matrices de co-occurrence ( $0^\circ$ ) :

$$P_{0,d}(i, j) = (x_1, y_1), (x_2, y_2) \in B \mid x_1 = x_2, |x_2 - x_1| = d, f(x_1, y_1) = i, f(x_2, y_2) = j$$

Quelques exemples de descripteurs (features) :

- Contraste :  $C(k, n) = \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} \frac{C_R(i, j)}{(i-j)^k}, i \neq j$
- Entropie :  $H = -\sum_{i=0}^{L-1} \sum_{j=0}^{L-1} C_R(i, j) \log_2 C_R(i, j)$

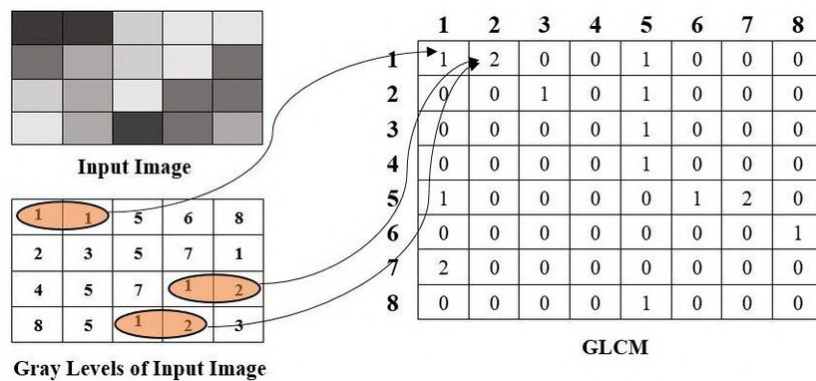


Figure 5 – Exemple d'une matrice de co-occurrence  $0^\circ$  en niveau de gris

L'entrée prendra donc la forme d'un tenseur  $3 \times 256 \times 256$ . Le modèle de deep learning s'appuiera sur le pipeline suivant :

1. 32 neurones  $3 \times 3$  convs
2. une couche ReLu
3. 32 neurones  $5 \times 5$  convs
4. une couche de max pooling
5. 64 neurones  $3 \times 3$  convs
6. une couche ReLu
7. 64 neurones  $5 \times 5$  convs
8. une couche de max pooling
9. 256 neurones dense
10. 256 neurones dense
11. une couche sigmoïde
12. un optimiseur variante du gradient stochastique

Le modèle utilisera le jeu de données CelebA qui sera converti en "DeepFake" par l'intermédiaire du logiciel StarGAN pour les négatifs. L'entraînement se fera avec 50% des données, la validation avec 25% et le test avec 25%. Le **batch size** sera de taille 40 avec un nombre d'**epoch** égale à 50.

### 3 Le réseau de neurones récurrents (LSTM)

Le réseau de neurones récurrents est utilisé pour l'analyse de séries temporelles, du langage (parole ou écrit) ainsi que la reconnaissance de formes. Il a une particularité qui lui permet de pouvoir se souvenir des événements précédents que ne peut pas faire un CNN par exemple

à cause de la disparition du gradient par la rétro-propagation qui a pour effet d'empêcher la modification des poids en fonction d'événements passés. La mémoire de la cellule est représentée par  $c_t$  qui représente la mémoire à long terme. Son évolution se fait par un réseau de 3 portes ; Forget Gate qui lui permet d'oublier de l'information, Input Gate qui permet d'ajouter de l'information et Output Gate qui permet de faire le lien entre les données de Input Gate et  $c_t$  afin de générer  $h_t$ .

Au niveau mathématique, les portes, la couche cachée  $h_t$  et l'état de la cellule s'écrivent :

$$F_t = \sigma(W_F x_t + U_F h_{t-1} + b_F) \text{ (forget gate)}$$

$$I_t = \sigma(W_I x_t + U_I h_{t-1} + b_I) \text{ (input gate)}$$

$$O_t = \sigma(W_O x_t + U_O h_{t-1} + b_O) \text{ (output gate)}$$

$$c_t = F_t \circ c_{t-1} + I_t \circ \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = O_t \circ \tanh(c_t)$$

$$o_t = f(W_o h_t + b_o)$$

où  $\sigma$  et  $\tanh$  sont des fonctions d'activation.

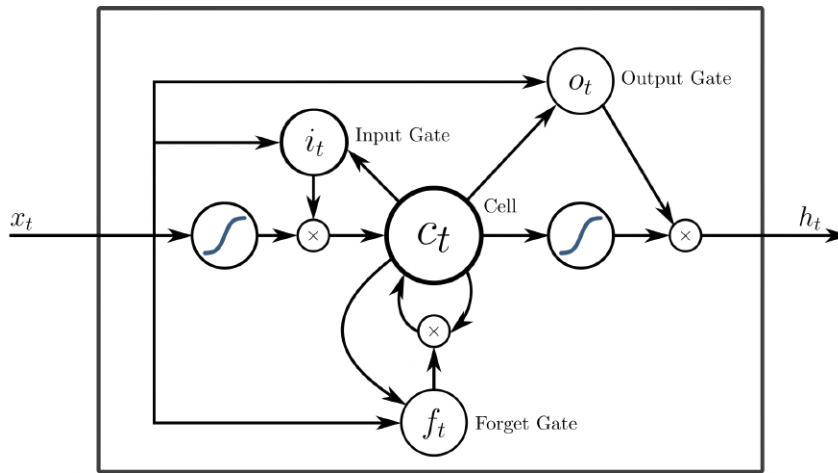


Figure 6 – Structure d'une couche LSTM

### 3.1 Conception ciblée sur la détection de l'auto-encodeur

Dans cette conception, un CNN est utilisé avec en complément un LSTM (appelé LSTM convolutionnel) afin de pouvoir bénéficier d'une mémoire à long terme (ce que ne peut pas avoir un CNN avec la disparition du gradient), ce qui permet ainsi de pouvoir détecter les incohérences dans des séquences temporelles. Le CNN va donc agir en faisant l'extraction des features qu'il aura reconnu comme étant pertinente, les fournir au LSTM qui fournira ensuite un descripteur de séquence. L'architecture de ce pipeline de deep learning est le suivant : Le CNN en entrée utilisera un modèle ImageNet pré-existant avec une architecture InceptionV3 [3]. Le LSTM recevra en entrée un vecteur de features de taille 2048 après une dernière phase de pooling du CNN. Le LSTM est suivi d'une couche de 512 neurones complètement connectées avec 0.5 en probabilité de "drop out" qui se termine par une couche softmax en sortie pour calculer les probabilités inhérent à la séquence en entrée. Il est nécessaire d'entraîner le modèle LSTM avec des exemples positifs et négatifs qui seront fournis par des logiciels de création de "DeepFake". Le nombre d'épochs du modèle est de 10.



## 4 Conception générale du système

Le programme de détection du contenu "DeepFake" s'appuiera sur plusieurs packages contenant les classes qui permettront d'accomplir les différentes fonctionnalités à travers l'objectif de la détection :

package : "analyze"

1. une classe "Analyze" qui réalisera l'analyse avec les modèles de deep learning implémentés et permettant d'avoir une confiance dans la falsification potentielle du contenu.

package : "train"

1. une classe "Train" qui permettra d'entraîner les différents modèles de deep learning, charger un modèle pré-entraîné ou le sauvegarder sous forme de fichier.
2. package "model"
  - (a) une classe "Model" qui contient les caractéristiques des modèles d'apprentissage profond qui seront utilisés pour la détection.

package : "extract"

1. une classe "Extract" qui permettra d'extraire les visages du contenu (image ou vidéo) pour les utiliser dans l'analyse. Son autre rôle "optionnel" est de permettre la création des exemples négatifs pour l'auto-encodeur sans avoir recours à un logiciel de "DeepFake".

package : "interface"

1. une classe "View" qui permettra à l'utilisateur d'utiliser le logiciel par le biais d'une interface graphique (front-end).
2. une classe "Controller" qui permettra à l'interface de communiquer avec le back-end.

L'interface utilisera la modélisation MVC implémenté par le biais de la classe "View" pour la vue, "Controller" pour le contrôleur et "Analyze" pour la partie modèle.

Une description plus exhaustive du diagramme de classes est à retrouver dans la partie E, "Cahier du développeur".



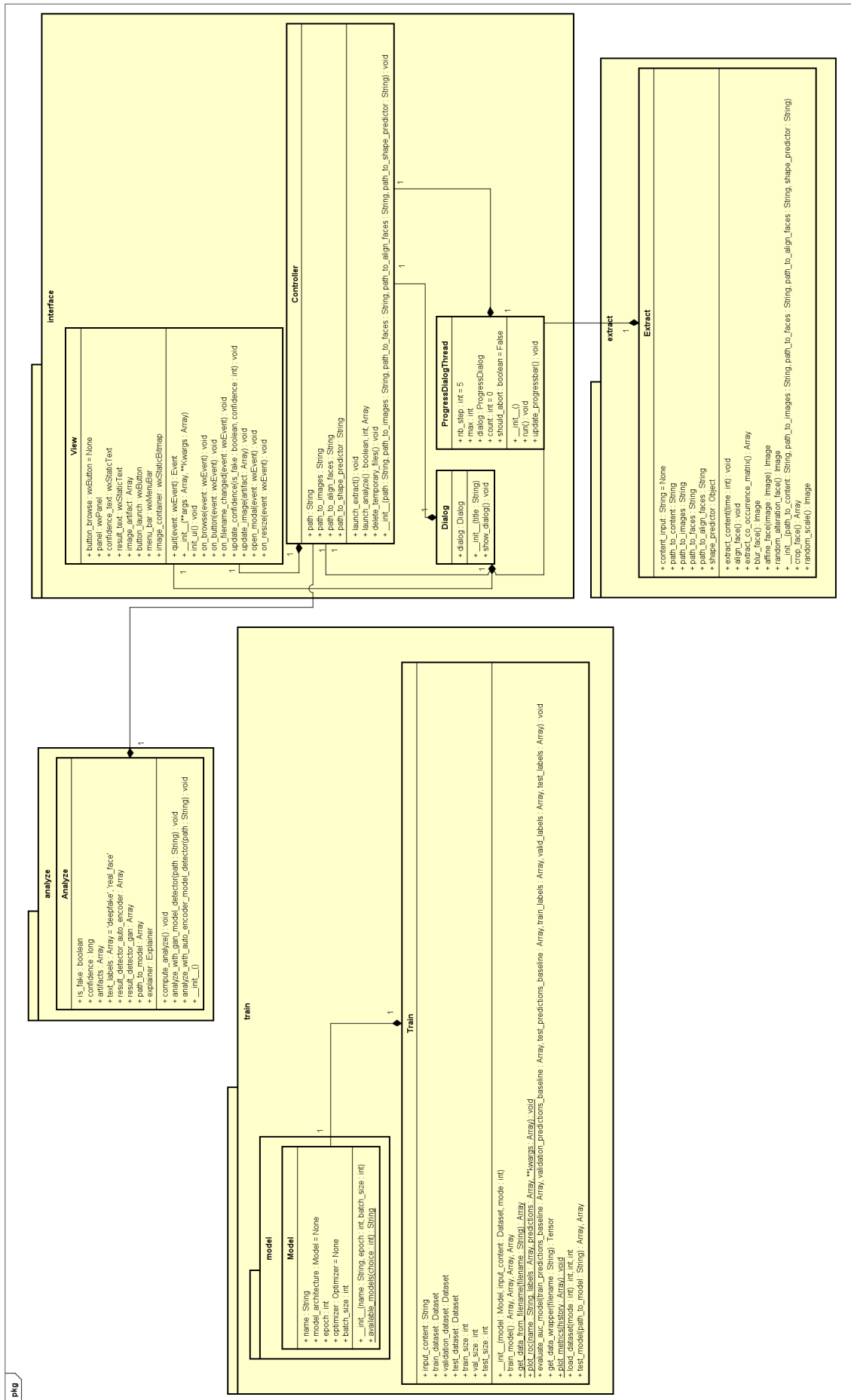


Figure 7 – Diagramme de classes du système

# 5

## Mise en œuvre

Ce chapitre permet de conclure sur les différentes actions mises en œuvre et résultats obtenus à la suite de l'implémentation des solutions retenues lors de l'état de l'art ainsi que lors de l'analyse préliminaire.

### 1 Pré-traitement des jeux de données

Avant de pouvoir entraîner et/ou tester un quelconque réseau de neurones, il convient de posséder au préalable un ensemble de jeux de données correcte afin d'obtenir des résultats et prédictions fiables utiles à l'étude. Pour cela, des pré-traitements sont appliqués sur chacune des images dont celle de faire ressortir uniquement les zones d'intérêt qui seront par ailleurs, décrits plus en détails dans les parties ci-dessous. Il est important de noter que les jeux de données disponibles sont en majorités sous forme de vidéo, ce dont les modèles de convolutions ne peuvent se satisfaire du fait que leur application s'exerce sur des images afin de créer des feature map (résultat de la convolution) et par conséquent une extraction est donc nécessaire afin d'obtenir les images (frames) qui les composent. Diverses extractions seront réalisées en amont par un script Python qui permet de sélectionner le pré-traitement souhaité; soit l'extraction des frames d'une vidéo et des visages présents dans l'image ou sinon leurs matrices de co-occurrence. Ci-dessous, divers exemples d'extractions possibles :



Figure 1 – Exemple d'une image extraite d'une vidéo falsifiée



**Figure 2** – Exemple d'un des nombreux visages extrait de l'image précédente

Afin de limiter le risque de surapprentissage par la multiplication des frames du même visage, une limitation du nombre d'images extraites est imposée (toute les  $s$  secondes), ce qui permet alors d'avoir une diversité non redondante dans les différentes expressions faciales dû à l'espacement temporel. La taille du jeu de données (en négatif et positif) doit être au moins égale à celle des papiers de recherche afin d'être au plus proche des conditions d'expérimentation lorsqu'il faudra comparer les résultats obtenus à ceux-ci. D'autre part, l'égalité est importante car c'est directement cette taille qui influence les mises à jours des poids et biais du modèle dans l'entraînement dont le but est qu'il devienne spécialisé à égalité sur les diverses classes en sortie.

L'exemple ci-dessous illustre l'extraction d'une des matrices de co-occurrence d'une image en couleur (RGB), composée de trois de ces matrices, soit une pour chaque canal rouge, vert et bleu. L'extraction de ces matrices peut, et se fait sur de multiples degrés, c'est à dire en  $0, \pi/2, \pi/4, 3\pi/4$  ( $0^\circ, 90^\circ, 45^\circ, 135^\circ$ ) et entre deux pixels voisins direct mais cela peut évidemment s'étendre à la distance de voisinage souhaitée. Il est alors possible ensuite d'en extraire des informations telle que l'entropie, le contraste, les moments, ... mais qui ne seront pas abordées dans le cadre de cette étude.

0	0	0	1	3	8	18	10	7	5
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	2	13	13	40	26	16
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	3	7	28	140	52
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	2	0	4	2	25	64	286

**Figure 3** – Extrait  $10 \times 10$  d'une matrice  $256 \times 256$  de co-occurrence de degré  $0^\circ$  à partir de la matrice de couleur rouge d'une image RGB

A la suite de l'extraction, la matrice obtenue est de taille  $4 \times 3 \times 256 \times 256$ , soit un ensemble de 4 matrices qui contiennent elles-mêmes trois matrices (trois canaux) de taille  $256 \times 256$  (niveaux de gris  $[0,255]$  de chaque couleur). Néanmoins, ce format est contraignant lorsque qu'il est

transformé sous une forme de tenseur. En effet, une image possède une forme  $L \times H \times 3$ , c'est donc le type de format attendu par un réseau de neurones convolutifs qui applique ensuite des masques de convolutions sur un espace  $\mathbf{R}^{W \times H \times 3}$  or sous cette forme l'espace correspond à  $\mathbf{R}^{3 \times 256 \times 256}$ . L'astuce consiste donc au préalable à changer l'espace de ces matrices dans le format voulu grâce à l'opération de transposé permettant d'inverser les colonnes et lignes donnant au final  $256 \times 256 \times 3 \times 4$ , soit 4 images créées dans l'espace  $\mathbf{R}^{256 \times 256 \times 3}$  à partir des matrices de co-occurrence rouge, vert et bleu.

## 1.1 Jeux de données utilisés pour la détection de DeepFake créés par auto-encodeur

Lors de l'état de l'art, un nombre important de jeux de données ont été identifiés en corrélation avec l'objectif de pouvoir détecter les falsifications créés au moyen de l'auto-encodeur. Il était aussi question de pouvoir générer des "Deepfake" si besoin grâce à différents logiciels tel que FaceSwap ou bien de les simuler par une opérations de floutage et de transformations affine. Ces deux options n'ont finalement pas été utilisées au vue du nombre important de datasets déjà disponibles, néanmoins la deuxième option a été implémentée au besoin dans le code.

Les jeux de données utilisés sont issues de FaceForensics++ [12] fourni par Google et qui compile un ensemble de "DeepFake" (1000 vidéos par technologie) créés par différents logiciels ou technologies utilisant l'auto-encodeur ainsi que DeepFake Detection Challenge fourni par Facebook ( $\approx 100000$  vidéos) [5] qui utilise les dernières méthodes de génération de "DeepFake" par auto-encodeur ou en association avec une autre technologie voire en modifiant uniquement la voix. Un jeu de données apparu très récemment (le plus large actuellement) Celeb-DF contient plus de 5000 vidéos de "DeepFake" néanmoins les procédés de création n'ayant pas été rendu publique, cela a conduit à ne pas le prendre en compte pour l'étude. L'autre ensemble de jeux de données est issu de ceux présentés dans les papiers de recherche, à savoir VidTIMIT [13], DeepfakeTIMIT [9], UADFV [15] utilisés pour la validation du modèle après entraînement afin de pouvoir alors comparer les résultats obtenus à ceux issus du papier de recherche où est majoritairement extraite l'architecture deep learning utilisé. Ces jeux de données permettent en outre de pouvoir avoir un aperçu assez large dans la création de "DeepFake" auto-encodeur (qualité de l'image, degré de réalisme, type d'auto-encodeur, ...). L'un des jeux de données présent dans FaceForensics++, dénommé DeepFake est composé de 1000 vidéos de  $\pm 10$  secondes provenant de Youtube (positives) et dont les 1000 vidéos "DeepFake" ont été générés grâce au logiciel FaceSwap, logiciel open-source le plus communément utilisé par sa simplicité d'utilisation et qui est donc à même d'être le plus répandu dans les créations existante.

Le nombre important de vidéos homogènes dans la technologie employée (uniquement sur l'image du visage) et la qualité du "DeepFake" équivalente à la première génération qui est utilisée dans le papier de recherche d'où est issu l'architecture du modèle, ont permis de retenir ce jeu de données, DeepFake, pour l'entraînement du modèle d'apprentissage profond.

## 1.2 Jeux de données utilisés pour la détection de DeepFake créés par réseau antagoniste génératif (GAN)

Le jeu de données utilisé dans le papier de recherche [11], CelebA en lien avec l'utilisation de StarGAN pour créer de nouveaux visages avec le réseau antagoniste génératif s'est avéré peu convainquant au rendu de certaines images avec le modèle pré-entraîné fourni du logiciel. De ce fait et par manque de temps pour générer un modèle plus performant, le jeu de données utilisé a finalement été celui fourni par Flickr (site de partage photos et vidéos) en haute qualité, de dimension  $1024 \times 1024$  dont sont issues les "DeepFake" créés par le GAN open-source de NVIDIA



**Figure 4** – Exemple d'une image venant du jeu de données DeepFake de FaceForensics++

(i.e. le plus avancé à ce jour) qui est implémenté avec l'architecture StyleGAN2 [8] (version 2 de StyleGAN, sortie en décembre 2019). Cela permet de respecter le même protocole que pour StarGAN, à savoir utiliser des images qui ont permis la génération de "DeepFake" créés par GAN.



**Figure 5** – Exemple d'une image créé par le logiciel StarGAN



**Figure 6** – Exemple d'images créés par le GAN StyleGAN2

## 2 Métriques de la performance des modèles d'apprentissage profond

Afin de pouvoir situer de manière fiable la performance des modèles implémentés, les métriques utilisées sur la base d'entraînement, de validation et les diverses bases de tests seront la perte (loss), la précision (accuracy) et la métrique AUROC qui permettront si possible la comparaison avec celles des papiers de recherche.

Afin de justifier le choix de ces métriques, une définition est donnée pour chacune d'entre elles ci-dessous :

- La **perte (loss)** est une métrique qui représente la performance du modèle dans la classification quantifiée par une fonction de perte et dont **l'entropie croisée binaire (2 classes) sera celle utilisée pour les différents modèles**. L'entropie croisée binaire se résume mathématiquement tel que :

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \log(p(y_i)) + (1 - y_i) \log(1 - p(y_i))$$

où  $y_i$  est l'étiquette (1 ou 0) et  $p(y_i)$  la probabilité

L'objectif recherché par la classification est d'obtenir  $H_p(q) = 0$  qui signifie un classificateur parfait soit le but avec les classes de l'étude. Néanmoins cette valeur est théorique et se vérifie par un résultat d'expérimentation qui est plutôt  $\approx 0$ .

Afin de réduire cette perte, de multiples optimiseurs sont possibles dont certains sont décrits ultérieurement lors des implémentations. Elle est calculée à chaque fin de batch et permet de visualiser les progrès du modèle (voir Figure 11). Cette métrique est étroitement liée avec la précision, l'optimisation de la solution permet en théorie que le modèle devienne plus performant dans ses prédictions.

- La **précision (accuracy)** est une métrique qui se base sur le nombre de cas positifs pour lequel le modèle prédit juste par rapport au nombre de cas présentés.

Elle est calculée à chaque fin de batch.

Sa formule est donnée tel que :  $Precision = \frac{VP}{VP+FP}$

où VP est le nombre de vrais positifs prédit et FP le nombre de faux positifs prédit.

- **AUROC** est une métrique qui se base sur deux calculs; AUC et ROC. Le ROC est une courbe qui représente le taux de vrais positifs en fonction du taux de faux positifs.

Taux de vrais positifs :  $TVP = \frac{VP}{VP+FN}$

Taux de faux positifs :  $TFP = \frac{FP}{FP+VN}$

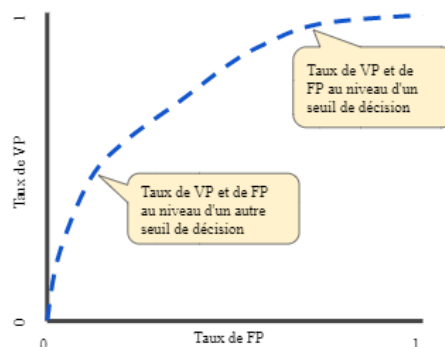


Figure 7 – Exemple d'une courbe ROC



Le **AUC (area under the curve)** est la mesure de l'aire sous la courbe ROC (probabilité que le modèle classe un exemple positif aléatoire au-dessus d'un exemple négatif aléatoire). La valeur est comprise entre  $[0, 1]$  avec 1 dans le cas d'une classification parfaite.

### 3 Implémentations du réseau de neurones ResNet50 pour la détection de DeepFake créés par auto-encodeur

Dans cette partie seront décrits les nombreux essais et implémentations du modèle de deep learning utilisant le modèle pré-entraîné ResNet50 du papier de recherche cité précédemment dans les solutions retenues.

Pour commencer, le nombre de classes est égal à 2 (Vrai / Faux), ce qui signifie qu'il faut en sortie avoir un à deux neurones pour la bonne représentation de celles-ci. Sachant qu'il est souhaitable de pouvoir comparer entre une prédiction du détecteur de "DeepFake" créés par GAN et une prédiction du détecteur de "DeepFake" créés par auto-encodeur, une sortie avec deux neurones est préférable car l'application d'une fonction d'activation softmax en sortie donne une prédiction tel que  $\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \forall j \in \{1, \dots, K\}$  c'est à dire que la somme des probabilités de chaque sortie est égale à 1 donc idéale pour comparer entre celles des deux modèles.

Néanmoins après plusieurs essais avec deux neurones en sortie et l'utilisation de softmax, la précision du modèle est restée constante à 0.5000, cela a donc obligé à ne pouvoir utiliser qu'un **seul neurone en sortie des modèles avec l'application d'une fonction d'activation sigmoïde**.

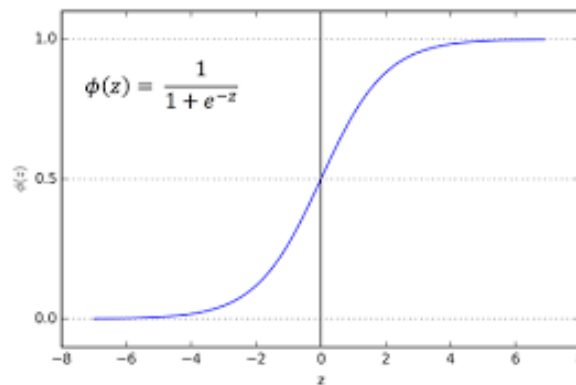


Figure 8 – Fonction d'activation sigmoïde

La fonction sigmoïde donne une prédiction entre  $[0, 1]$  pour chaque sortie avec une frontière entre les deux classes symbolisé par 0.5.

L'utilisation d'un modèle pré-entraîné entraîne généralement une modification de celui-ci en sortie car son entraînement est d'une part éloigné des classes de l'étude (entraînement avec de multiples catégories d'images en dimension  $224 \times 224 \times 3$ ) et d'autre part, le nombre de neurones en sortie qui est de 1000 doit être ramené à 2. Il faut donc déconnecter la partie de décision initiale du modèle qui permet d'obtenir uniquement en sortie la dernière couche de convolution, ce qui permet l'ajout d'autres couches ensuite. La dernière couche de convolution étant une feature map  $\mathbf{R}^{W \times H}$ , celle-ci n'est pas exploitable sous cette forme pour faire la prédiction en sortie, il faut donc dans ce cas faire à la suite un flatten (aplatissement) qui permet de mettre sous forme vectorielle la feature map représenté par :

$$N_{conv} \times dimension_{image} \times hauteur_{image} \times largeur_{image} \\ \text{vers un vecteur : } N_{conv} \times (dimension_{image} \times hauteur_{image} \times largeur_{image})$$

Le résultat est ensuite passé à la couche complètement connectée (fully connected) représentée par la couche de décision, soit le neurone en sortie avec la fonction d'activation qui permet de faire la classification.

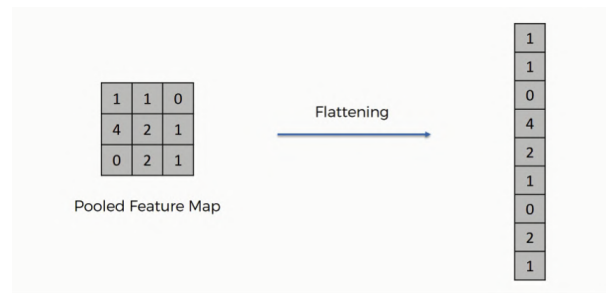


Figure 9 – Exemple d'un aplatissement après un résultat de convolution

L'implémentation du modèle est néanmoins sujette à des interprétations vis-à-vis du papier de recherche notamment le nombre de couches en fine-tuning (dégeler les poids des couches du modèle pré-entraîné pour augmenter la spécialisation sur les images des classes). Par ailleurs, afin de garder les performances du modèle tout au long des epochs, un EarlyStopping (arrêt en avance) est mis en place et permet de sauvegarder les nouveaux poids uniquement si ceux-ci donnent de meilleurs résultats qu'à l'époque précédente tout en stoppant le modèle si celui-ci stagne (au bout de 10 epochs). L'optimiseur est celui extrait du papier de recherche, le SGD (algorithme de descente de gradient stochastique) implémenté avec un scheduler (planification) en décroissance exponentielle possédant un taux d'apprentissage  $\eta = 0.001$  initial et qui applique ensuite une décroissance du taux d'apprentissage  $\eta$  tout les 1000 pas, à un pourcentage de 95%.

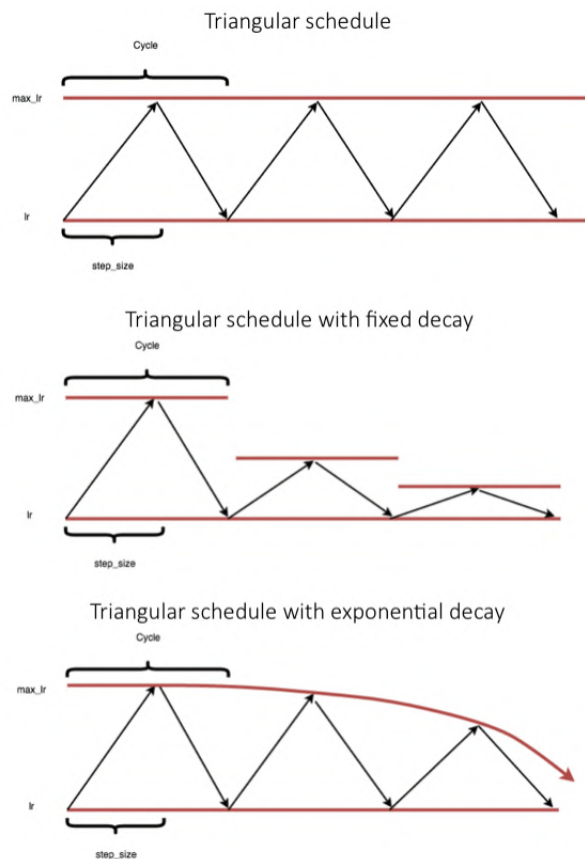
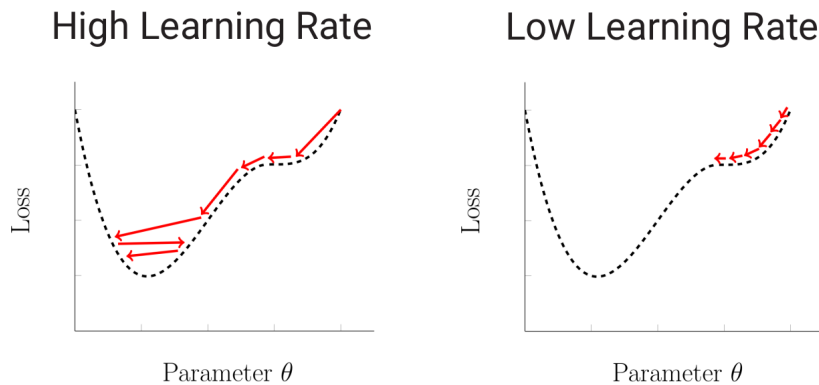


Figure 10 – Exemples de scheduler triangulaire



La décroissance exponentielle sert à appliquer une décroissance mesurée du taux d'apprentissage afin de diminuer le risque de ne pas converger vers le minimum global ou sinon vers un minimum local. En effet, le problème est celui de l'optimisation de la solution d'une fonction objectif qui n'est pas forcément de forme convexe et où un algorithme simple de descente de gradient peut ne pas être suffisant pour s'assurer de la convergence vers le minimum global.



**Figure 11** – Exemple de la différence entre un taux d'apprentissage important ou faible

L'objectif est donc la minimisation de la fonction objectif par la descente de gradient stochastique dont l'équation mathématique est représentée ci-dessous :

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^n \nabla Q_i(w)$$

où  $w$  est le vecteur de paramètres,  $Q_i(w)$  la valeur de la fonction objectif du  $i$ ème exemple,  $Q(w)$  le risque empirique et  $\eta$  est le taux d'apprentissage.

Les différents essais dans l'implémentation du modèle sont listés ci-dessous :  
Architecture commune des essais du modèle :

Model : "sequential"		
Layer (type)	Output Shape	Param #
resnet50 (Model)	(None, 7, 7, 2048)	23587712

Le modèle ResNet50 [14] étant composé de 175 couches, le nombre de paramètres potentiellement dégelable est de 23 587 712. La couche de décision étant supprimée, la dernière couche est celle de convolution 7x7 avec 2048 neurones en activation.

Modifications apportés à l'architecture du modèle de base :

1. Flatten, Dense(1, 'sigmoid') sans fine-tuning sur l'ensemble des couches et optimiseur SGD avec un taux d'apprentissage  $\eta$  égal à 0.001
2. Flatten, Dense(1, 'sigmoid') avec fine-tuning sur l'ensemble des couches et optimiseur SGD avec un taux d'apprentissage  $\eta$  égal à 0.001
3. Flatten, Dense(1, 'sigmoid') avec fine-tuning sur l'ensemble des couches et optimiseur SGD avec un taux d'apprentissage  $\eta$  égal à 0.0001

Les différents entraînements, validations et tests seront décrits dans la partie suivante concernée.

## 4 Implémentations du réseau de neurones CNN pour la détection de DeepFake créés par GAN

Dans cette partie seront décrits les nombreux essais et implémentations du modèle de deep learning utilisant le modèle CNN du papier de recherche cité précédemment dans les solutions retenues.

L'implémentation du modèle est cette fois-ci beaucoup plus "flou" en ce qui concerne des paramètres importants tel que l'optimiseur utilisé, le taux d'apprentissage et l'ingestion des tenseurs des matrices de co-occurrence en entrée. Le seul indice sur l'optimiseur est qu'il s'agit d'une variante du gradient stochastique (SGD).

Plusieurs alternatives sont possibles avec la librairie Keras, le choix a donc été d'essayer avec l'optimiseur AdaGrad et son amélioration AdaDelta qui adaptent le taux d'apprentissage en fonction des paramètres  $\theta_d$ , utile dans ce cas car le taux d'apprentissage  $\eta$  utilisé dans le papier de recherche n'est pas connu.

1. L'optimiseur AdaGrad (adaptive gradient) est un algorithme de descente de gradient qui adapte le taux d'apprentissage pour chaque paramètre  $\theta_d$  en fonction des mises à jour plus ou moins fréquentes du gradient de celui-ci.

Le taux d'apprentissage  $\eta$  est alors multiplié par les éléments du vecteur  $G_{j,j}$ . De plus, la convergence est plus rapide que pour le SGD et il n'est pas très sensible au taux d'apprentissage initial. Les matrices  $G$  et  $G_{j,j}$  sont obtenues par :

$$G = \sum_{\tau=1}^t g_{\tau} g_{\tau}^T \text{ où } g_{\tau} = \nabla Q_i(w) \text{ le gradient à l'étape } \tau \text{ et } G_{j,j} = \sum_{\tau=1}^t g_{\tau,j}^2$$

En accumulant les amplitudes carrées du gradient lors de la descente, les paramètres sont alors mis à jour selon :

$$w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j$$

2. L'optimiseur AdaDelta (adaptive learning rate method) est un algorithme de descente de gradient qui reprend le concept de AdaGrad en adaptant le taux d'apprentissage pour chaque paramètre  $\theta_d$  mais sur une fenêtre de mise à jour du gradient corrigeant alors le défaut de AdaGrad qui voit son taux d'apprentissage devenir proche de zéro par l'accumulation du gradient et ne permet donc plus d'apprendre. De plus, il n'est plus nécessaire de définir un taux d'apprentissage  $\eta$  manuellement.

Les équations de l'optimiseur AdaDelta sont alors :

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1 - \rho) g_t^2$$

où  $E[g^2]_t$  est la moyenne du gradient au temps  $t$

$$\text{Cela devient ensuite : } \text{RMS}[g]_t = \rho \sqrt{E[g^2]_t} + \epsilon$$

$$\text{La mise à jour des paramètres se fait ensuite selon : } \Delta x_t = - \frac{\text{RMS}[\Delta x]_{t-1}}{\text{RMS}[g]_t} g_t$$

Les différents essais dans l'implémentation du modèle sont listés ci-dessous :

Architecture commune des essais du modèle :

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	896

activation (Activation)	(None, 254, 254, 32)	0
conv2d_1 (Conv2D)	(None, 250, 250, 32)	25632
max_pooling2d (MaxPooling2D)	(None, 125, 125, 32)	0
conv2d_2 (Conv2D)	(None, 123, 123, 64)	18496
activation_1 (Activation)	(None, 123, 123, 64)	0
conv2d_3 (Conv2D)	(None, 119, 119, 64)	102464
max_pooling2d_1 (MaxPooling2D)	(None, 59, 59, 64)	0
conv2d_4 (Conv2D)	(None, 57, 57, 128)	73856
activation_2 (Activation)	(None, 57, 57, 128)	0
conv2d_5 (Conv2D)	(None, 53, 53, 128)	409728
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
dense (Dense)	(None, 26, 26, 256)	33024
dense_1 (Dense)	(None, 26, 26, 256)	65792
flatten (Flatten)	(None, 173056)	0
dense_2 (Dense)	(None, 1)	173057
activation_3 (Activation)	(None, 1)	0
=====		
Total params: 902,945		
Trainable params: 902,945		

D'autre part, s'agissant d'un modèle conçu spécifiquement, l'ensemble des poids et biais sont dégelés. Le nombre de paramètres entraînable est de 902 945.

Modifications apportées à l'architecture du modèle de base :

1. Optimiseur AdaGrad avec un taux d'apprentissage  $\eta$  égal à 0.0001
2. Optimiseur AdaDelta avec un taux d'apprentissage  $\eta$  égal à 1.0 et  $\rho$  égal à 0.95 (paramètre par défaut)

Les différents entraînements, validations et tests seront décrits dans la partie suivante concernée.

## 5

## Implémentations du réseau de neurones CNN-LSTM pour la détection de DeepFake créés par auto-encodeur

Dans cette partie seront décrits les nombreux essais et implémentations du modèle de deep learning utilisant le modèle CNN-LSTM du papier de recherche cité précédemment dans les solutions retenues.

L'implémentation du modèle est décrit de façon sommaire néanmoins l'implémentation reste simple avec l'utilisation de InceptionV3 (sans fine-tune) en extracteur de features et un LSTM en sortie pour ajouter une analyse temporelle. Ce modèle est intéressant car contrairement aux autres il intègre une dimension temporelle utile lorsque la plupart des "DeepFake" sont sous forme vidéo. L'optimiseur utilisé est Adam, optimiseur usuel dans les réseaux de neurones et

qui consiste à reprendre les avantages de AdaGrad et RMSProp. En effet, il s'agit d'une méthode adaptative du gradient qui calcule les taux d'apprentissage pour les différents paramètres. Pour cela, il calcule les moyennes des gradients passés ainsi que leur carré tel que :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Pour contrer le fait que  $m_t$  et  $v_t$  soit proche de 0 lors de l'initialisation comme le taux de diminution  $\beta_1$  et  $\beta_2$ ,  $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$  et  $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$  sont utilisés pour corriger ces biais. Ainsi les paramètres sont mis à jour selon :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

L'architecture du modèle issu du papier de recherche n'a pas été modifiée et est visible ci-dessous :

Model: "model"

Layer (type)	Output Shape	Param #
=====		
image_input (InputLayer)	[(None, 20, 299, 299, 3)]	0
=====		
time_distributed (TimeDistri	(None, 20, 2048)	21802784
=====		
lstm (LSTM)	(None, 2048)	33562624
=====		
dense (Dense)	(None, 512)	1049088
=====		
dropout (Dropout)	(None, 512)	0
=====		
dense_1 (Dense)	(None, 512)	262656
=====		
dropout_1 (Dropout)	(None, 512)	0
=====		
dense_2 (Dense)	(None, 1)	513
=====		
Total params: 56,677,665		
Trainable params: 34,874,881		
Non-trainable params: 21,802,784		

Néanmoins, une erreur non identifiée lors de la sauvegarde du modèle n'a pas permis de pouvoir entraîner correctement celui-ci et en conséquence, ce modèle ne sera pas pris en compte dans l'étude.

## 6 Protocole d'entraînement du réseau de neurones ResNet50

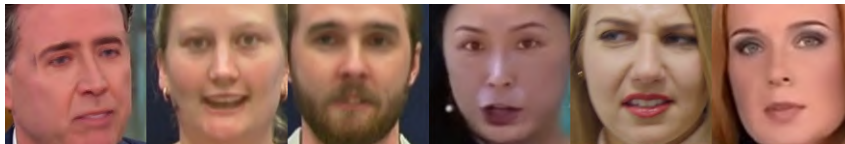
Cette section résume les différents entraînements, validations et tests du modèle ResNet50 avec les différents essais effectués en utilisant le jeu de données "DeepFakes" de FaceForensics++. Le jeu de données possède 1000 vidéos avec une qualité 720p (positives et négatives) qui représente des individus dans diverses émissions de télévision ou devant une webcam. La première étape est de répartir ce jeu de données en 3 répertoires distincts, entraînement, validation et test. La répartition du jeu de données n'étant pas mentionnée dans le papier de recherche, une séparation arbitraire devra être faite. Le modèle est entraîné sur 20 epochs avec une taille de batch de 64. Avec la configuration utilisée pour l'entraînement, chaque epoch prend  $\pm 45$  minutes avec  $\pm 15$  minutes pour la validation et sauvegarde du modèle entre les epochs. Le

modèle ne sera uniquement sauvegardé que dans le cas où sa précision augmente sur le jeu de données en validation par rapport à l'époque précédente. Afin de garder les performances du modèle tout au long des époques, un EarlyStopping (arrêt en avance) est mis en place et permet de sauvegarder les nouveaux poids uniquement si ceux-ci donnent de meilleurs résultats qu'à l'époque précédente tout en stoppant le modèle si celui-ci stagne (au bout de N époques).

Par ailleurs, lors du passage dans le pipeline, l'ensemble des images est normé sur 255 afin d'aider le réseau de neurones à apprendre plus rapidement car la donnée est alors centrée et le gradient agit donc uniformément sur chaque canal de l'image.

Malheureusement le papier de recherche ne fournit aucune de ses courbes (perte et précision) sur la base d'entraînement, de validation ou de test, il faudra donc se référer à la section des tests du modèle afin d'avoir des éléments de comparaison notamment avec la méthode AUROC expliquée précédemment.

Un échantillon ci-dessous permet de comparer les différentes qualités de "DeepFake" issues des jeux de données de l'étude (entraînement et tests) :



**Figure 12** – Échantillon d'images avec de gauche à droite, UADFV, DeepFakeTIMIT HQ, DeepFakeTIMIT LQ, Facebook Challenge, DeepFake Detection, NeuralTextures

Essai n°1 **L'essai est réalisé avec le modèle ResNet50,  $\eta = 0.001$  et sans fine-tune.**

La taille du jeu de données est de 38240 images (50% positives et 50% négatives) avec une modification de celui-ci par un pré-traitement qui consiste en un découpage des frames toute les 1 secondes suivi d'un redimensionnement centré sur le visage. Le dimensionnement des bases est de 80% pour l'entraînement et 20% pour le reste. Les bases sont donc composés respectivement de 30592 images et 3824 images (validation et test). Les jeux de données sont mélangés 1000 fois pour obtenir une alternance aléatoire entre les deux classes.

L'entraînement prend environ 45 minutes  $\times$  20 epochs soit 15 heures (13 epochs effective). Au terme de l'entraînement, les résultats obtenus sont les suivants :

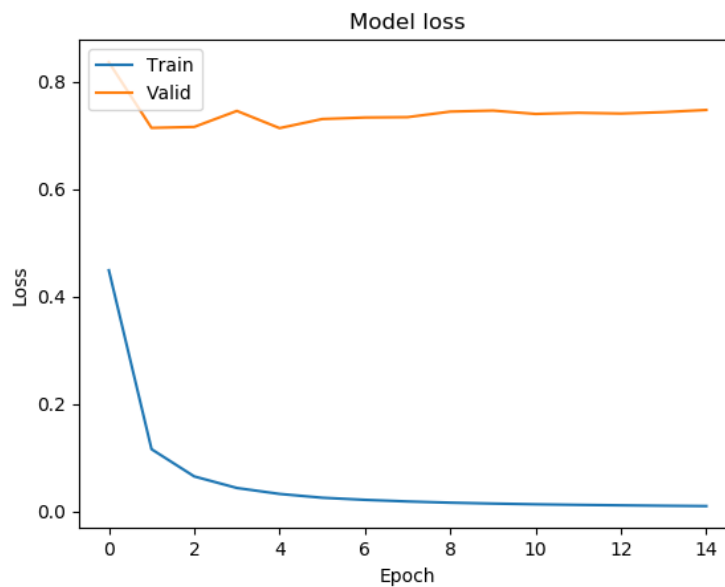


Figure 13 – Courbes de perte (loss) sur la base d'entraînement et de validation

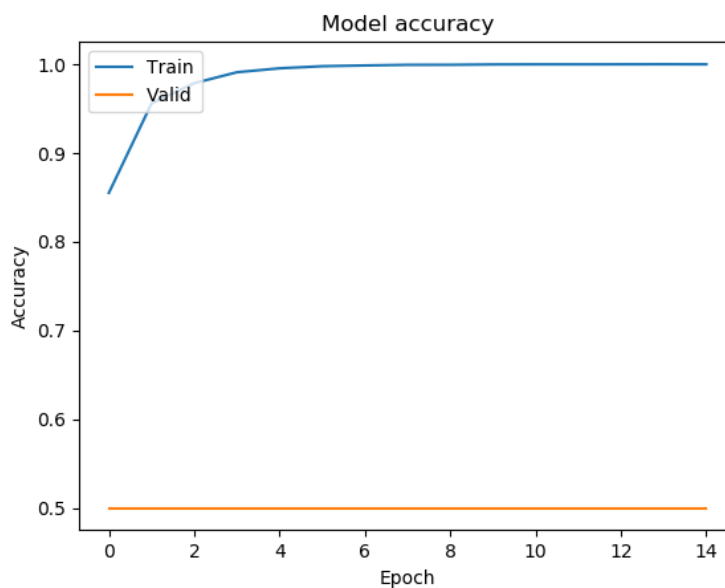


Figure 14 – Courbes de précision (accuracy) sur la base d'entraînement et de validation

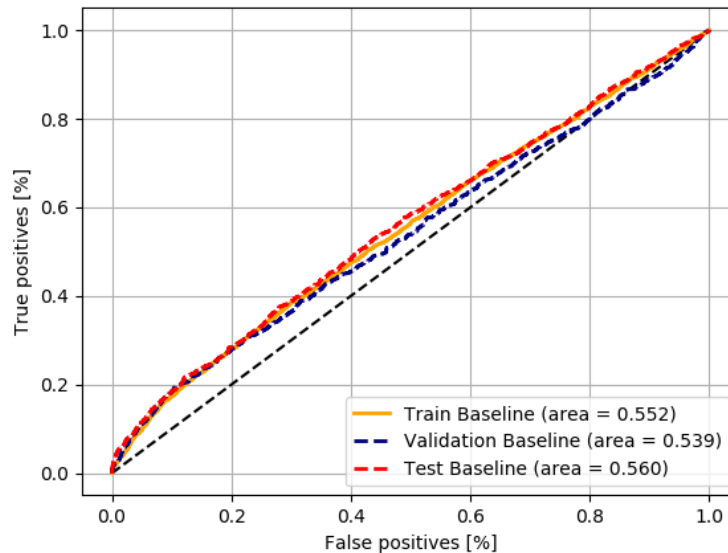


Figure 15 – Courbe ROC sur la base d'entraînement, validation et test

L'observation des valeurs de perte sur la base d'entraînement permet de constater qu'à partir de la 5ème epoch, la valeur est constante à 0.0248 alors que celle sur la base de validation est constante à 9.3228. Cela démontre un problème de surapprentissage (overfitting) où la base d'apprentissage est très bien apprise alors que le modèle n'arrive pas à généraliser sur une base indépendante.

Les courbes de précision confirment le problème avec une prédiction de 0.9972 sur la base d'entraînement à partir de la 5ème epoch alors que celle-ci est constante à 0.4995 sur la base de validation pour l'ensemble des epochs.

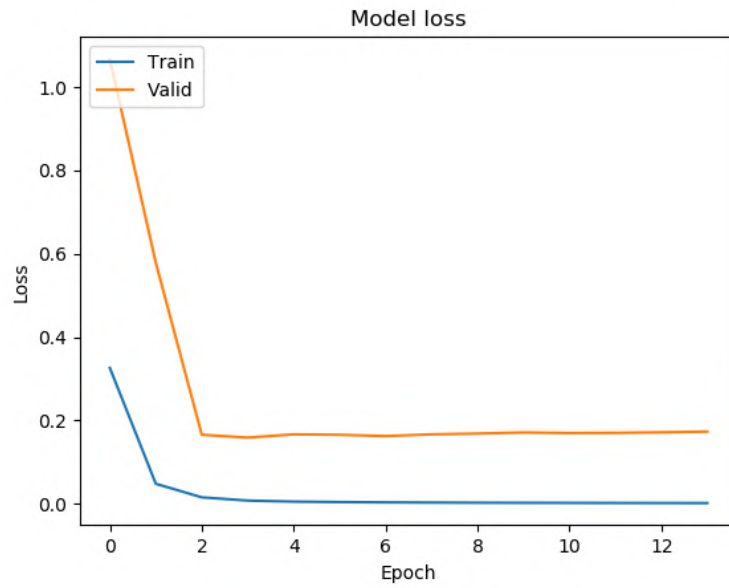
La perte sur la base de test est de 9.2676 et la précision 0.5002, ce qui correspond au même problème que sur la base de validation. Les courbes ROC démontrent le même effet précédemment évoqué, le modèle est par conséquent incapable de classer correctement les exemples sur les diverses bases.

Dans le prochain essai, un fine-tuning sera effectué sur l'ensemble des couches du réseau de neurones, ce qui devrait théoriquement améliorer les différents résultats.

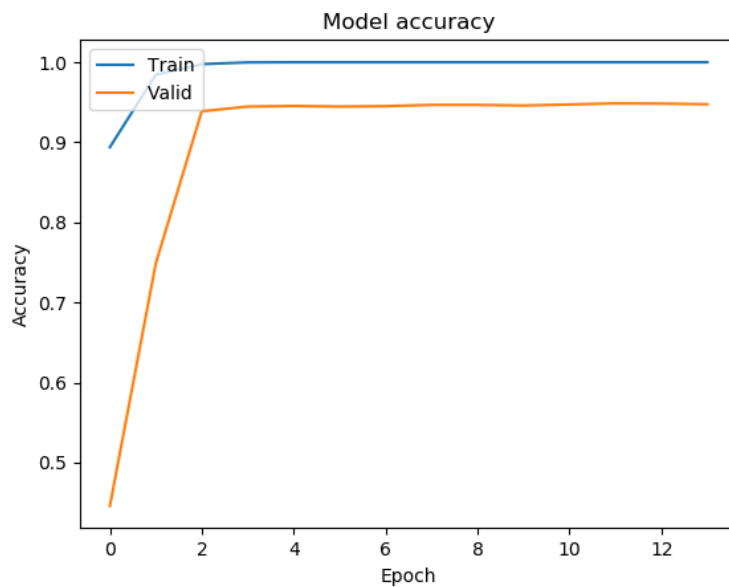
#### Essai n°2 **L'essai est réalisé avec le modèle ResNet50, $\eta = 0.001$ et fine-tune complet.**

La taille du jeu de données est de 38240 images (50% positives et 50% négatives) avec une modification de celui-ci par un pré-traitement qui consiste en un découpage des frames toute les 1 secondes suivi d'un redimensionnement centré sur le visage. Le dimensionnement des bases est de 80% pour l'entraînement et 20% pour le reste. Les bases sont donc composés respectivement de 30592 images et 3824 images (validation et test). Les jeux de données sont mélangés 1000 fois pour obtenir une alternance aléatoire entre les deux classes.

L'entraînement prend environ 45 minutes  $\times$  20 epochs soit 15 heures (13 epochs effective). Au terme de l'entraînement, les résultats obtenus sont les suivants :

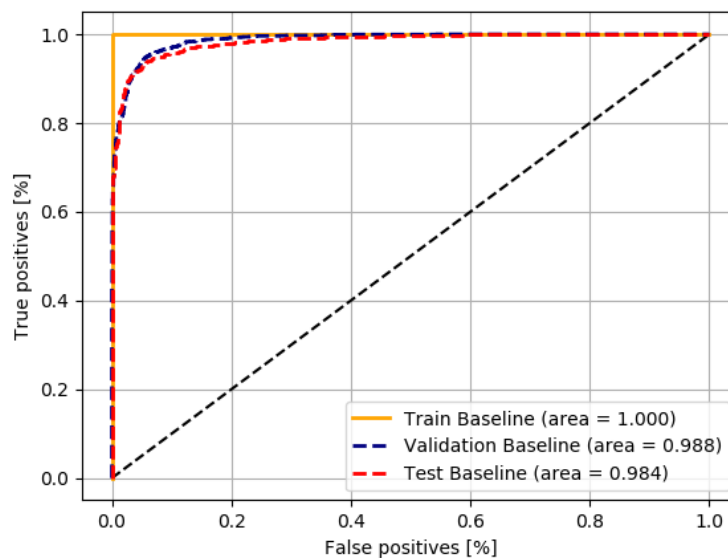


**Figure 16** – Courbes de la perte du modèle au cours des epochs sur la base d'entraînement et validation



**Figure 17** – Courbes de précision (accuracy) sur la base d'entraînement et de validation





**Figure 18** – Courbe ROC sur la base d'entraînement, validation et test

L'observations des courbes de perte (loss) montrent que l'apprentissage sur la base d'entraînement atteint à la 2ème epoch sa plus faible valeur 0.0015 alors que celle sur la base de validation atteint 0.02309. Cela démontre que le modèle performe très bien cette fois-ci lorsqu'il faut décider entre les deux classes.

Suivant la logique de la courbe de perte, la précision (accuracy) est à son maximum au terme de la 2ème epoch avec 1.0 sur la base d'entraînement et 0.9330 sur la base de validation.

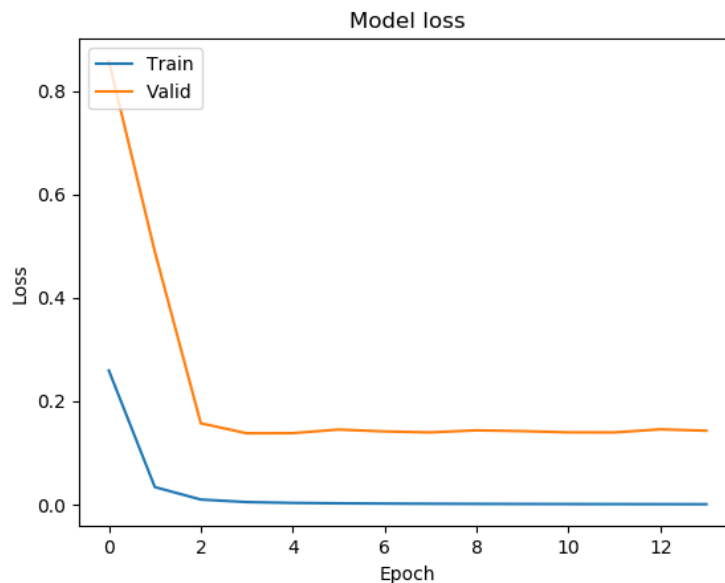
La perte sur la base de test est de 0.024 et sa précision de 0.936, ce qui confirme que l'apprentissage de la classification des "DeepFake" créés par auto-encodeur est un succès. L'entraînement est donc arrêté au bout de 13 epochs (early stopping) car la précision stagne ensuite. Par rapport à l'essai 1, le fine-tuning a permis au modèle pré-entraîné ResNet50 de "calibrer" ses poids en fonction des nouvelles classes.

Sachant que la performance de précision est supérieure à 90% sur les bases du modèle de l'essai, l'amélioration potentielle peut être l'augmentation de la donnée avec une taille de jeu de données plus important en prenant par exemple une image toute les 0.5 secondes plutôt que 1 fois par seconde.

Essai n°3 **L'essai est réalisé avec le modèle ResNet50,  $\eta = 0.001$  et fine-tune complet.**

La taille du jeu de données est de 47609 images soit 9369 images de plus que l'essai 1 (50% positives et 50% négatives) avec une modification de celui-ci par un pré-traitement qui consiste en un découpage des frames toute les 0.5 secondes suivi d'un redimensionnement centré sur le visage. Le dimensionnement des bases est de  $\approx 85\%$  pour l'entraînement et  $\approx 15\%$  pour le reste. Les bases sont donc composés respectivement de 40467 images et 3570 images (validation et test). Les jeux de données sont mélangés 1000 fois pour obtenir une alternance aléatoire entre les deux classes.

L'entraînement prend environ 45 minutes  $\times$  20 epochs soit 15 heures (13 epochs effective). Au terme de l'entraînement, les résultats obtenus sont les suivants :



**Figure 19** – Courbes de la perte du modèle au cours des epochs sur la base d'entraînement et validation

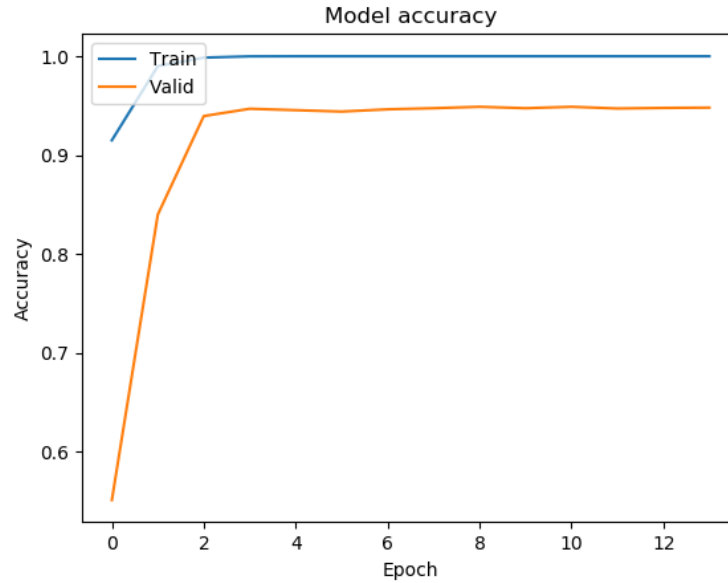


Figure 20 – Courbes de précision (accuracy) sur la base d'entraînement et de validation

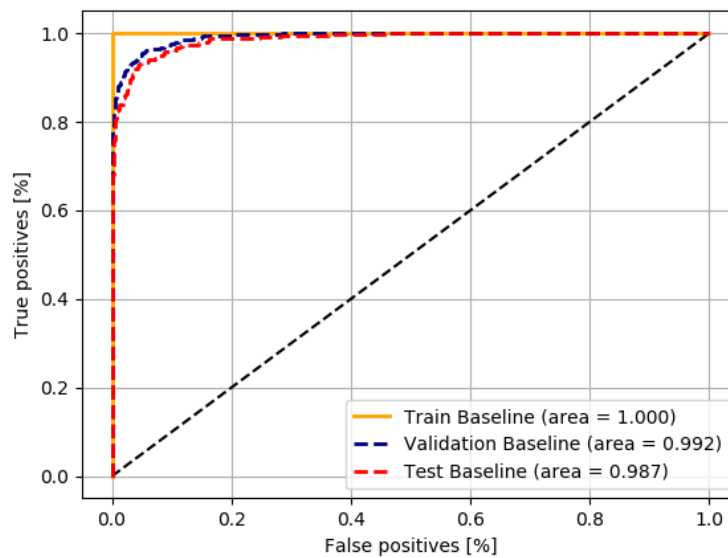


Figure 21 – Courbe ROC sur la base d'entraînement, validation et test

Les courbes ci-dessus sont quasiment identiques à celles de l'essai précédent et terminent à leur minimum et maximum respectif à la deuxième epoch. La conclusion à tirer de cela est que **l'augmentation du nombre d'images avec les mêmes visages n'a finalement aucune incidence** sur la perte ou la précision par rapport aux nombre d'epochs. Cet essai n'est donc pas à retenir dans l'amélioration des performances du modèle.

Essai n°4 **L'essai est réalisé avec le modèle ResNet50,  $\eta = 0.0001$  et fine-tune complet.**

La taille du jeu de données est de 38240 images (50% positives et 50% négatives) avec une modification de celui-ci par un pré-traitement qui consiste en un découpage des frames toute les 1 secondes suivi d'un redimensionnement centré sur le visage. Le dimensionnement des bases est de 80% pour l'entraînement et 20% pour le reste. Les bases sont donc composés respectivement de 30592 images et 3824 images (validation et test). Les jeux de données sont mélangés 1000 fois pour obtenir une alternance aléatoire entre les deux classes.

L'entraînement prend environ 45 minutes  $\times$  20 epochs soit 15 heures (18 epochs effective). Au terme de l'entraînement, les résultats obtenus sont les suivants :

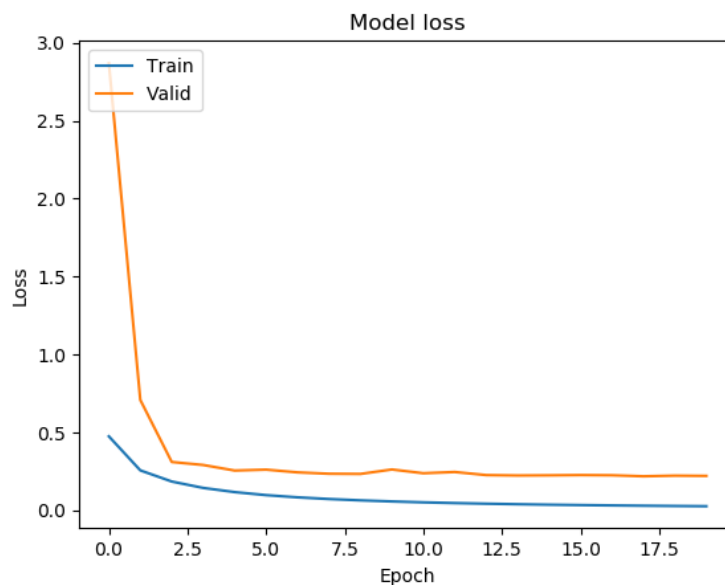


Figure 22 – Courbes de la perte du modèle au cours des epochs sur la base d'entraînement et validation

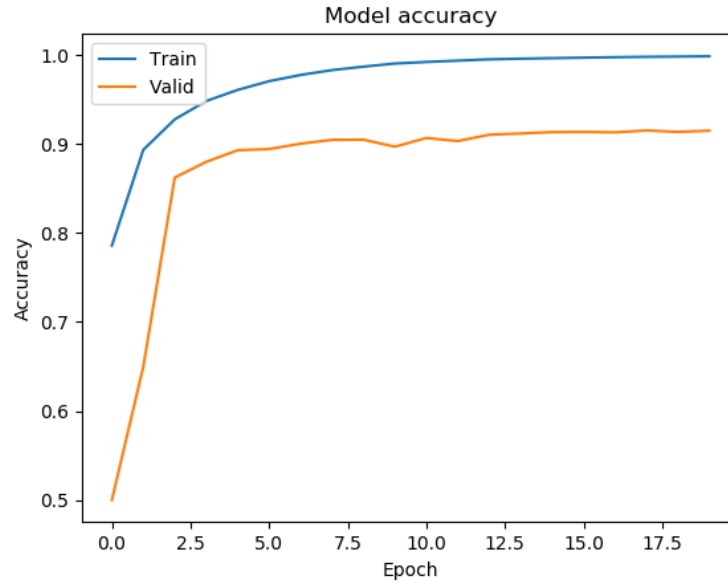


Figure 23 – Courbes de précision (accuracy) sur la base d'entraînement et de validation

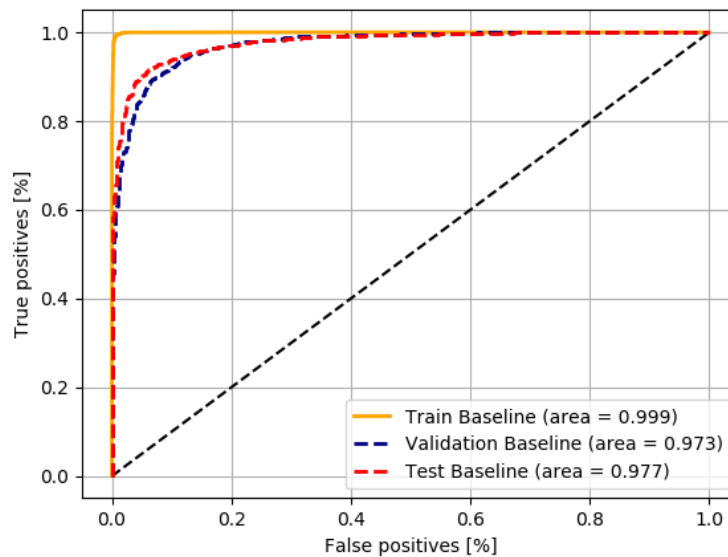


Figure 24 – Courbe ROC sur la base d'entraînement, validation et test

Les courbes de perte sur la base d'entraînement et de validation montrent que la réduction du taux d'apprentissage  $\eta$  de  $\frac{1}{10}$  par rapport au taux  $\eta$  des essais précédent entraîne le modèle à mettre plus de temps pour atteindre son minimum, qui est trouvé à la 4ème epoch au lieu de la 2ème dans les autres essais. La perte est de 0.027 en entraînement et de 0.22 en validation. Cela est supérieur à l'essai 2 où la perte était de 0.0015 sur la base d'entraînement et de 0.02309 sur la base de validation.

Les courbes de précision montrent aussi une plus lente augmentation avec 18 epochs pour atteindre le maximum de 0.9986 en entraînement et 0.9150 en validation. Cette dernière valeur est inférieure de 0.150 en précision par rapport à celle de l'essai 2. Les scores AUC sont d'ailleurs en légère diminution d'un dixième par rapport à l'essai 2. Une question se pose alors de la nécessité de faire 20 epochs par rapport au papier de recherche. En effet, vis-à-vis des courbes obtenus, cela est inutile néanmoins en relativisant le nombre de

visages qui est très largement inférieur à celui du papier de recherche (1000 visages contre 24442 visages), la courbe de perte est possiblement (voire significativement) différente par rapport à celles des essais du papier de recherche.

**Le modèle retenu sera donc celui de l'essai 2 au vue de ses performances supérieures aux autres essais.**

Les tableaux ci-dessous permettent de résumer les performances du modèle sélectionné :

Modèle - Base d'entraînement	Perte	Précision	AUC
ResNet50, $\eta = 0.001$ et fine-tune complet	0.0015	1.0	1.0

Modèle - Base de validation	Perte	Précision	AUC
ResNet50, $\eta = 0.001$ et fine-tune complet	0.02309	0.9330	0.988

Modèle - Base de test	Perte	Précision	AUC
ResNet50, $\eta = 0.001$ et fine-tune complet	0.024	0.936	0.984

## 7 Protocole de tests du réseau de neurones ResNet50 après entraînement

Dans cette section seront notamment décrits les tests effectués sur le modèle sélectionné.

- Comparaison du test sur le jeu de données UADFV avec le modèle entraîné (33644 images dimensionnées en prenant le visage uniquement) et le modèle du papier de recherche (32752 images, l'intervalle d'extraction étant certainement différent) :

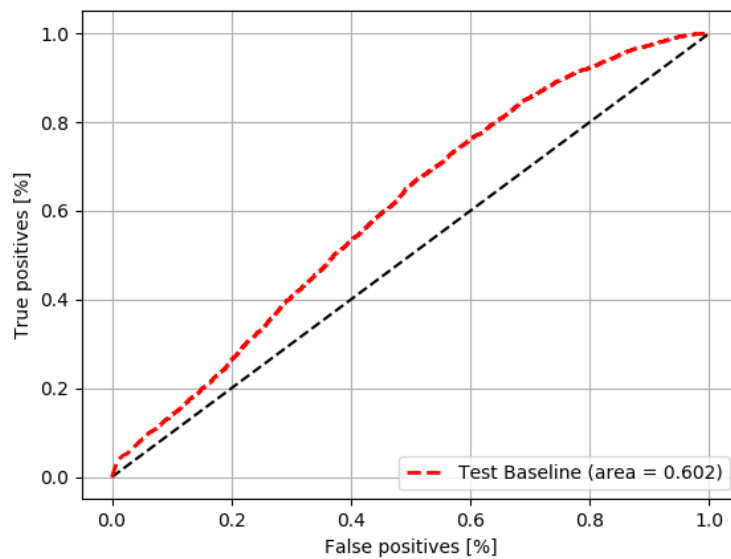


Figure 25 – Courbe ROC du modèle sur le jeu de données UADFV

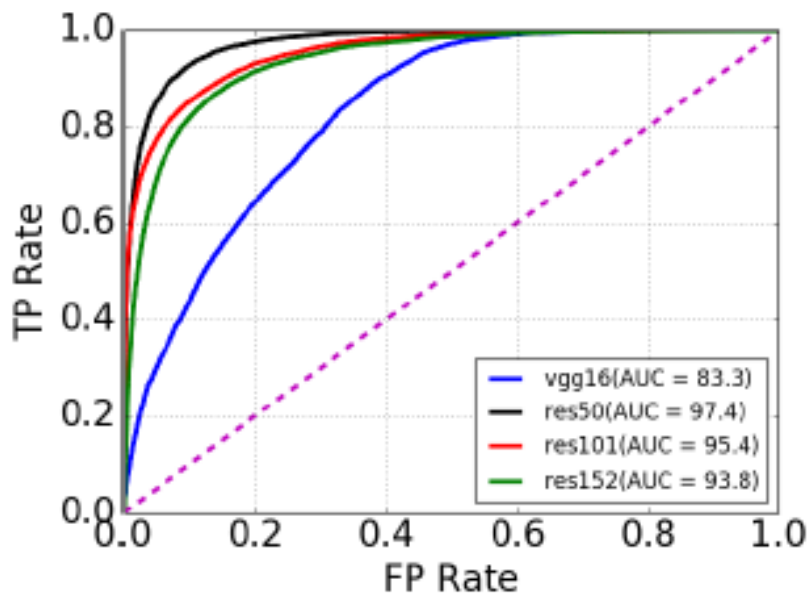


Figure 26 – Courbe ROC du modèle issu du papier de recherche sur le jeu de données UADFV

Le résultat de la performance sur UADFV est mauvais avec seulement de 60.2% (un peu mieux que le hasard) comparé au résultat du papier de recherche qui est de 97.4%. Ce score est certainement dû au fait d'avoir un modèle trop spécialisé sur une technologie particulière (FaceSwap). En effet, UADFV est composé de "DeepFake" créés par FakeApp, une application mobile dont le résultat est beaucoup plus avancé que FaceSwap d'où le fait que le modèle peine à reconnaître ceux-ci.

- Comparaison du test sur le jeu de données VidTIMIT/DeepFakeTIMIT HQ avec le modèle entraîné (34023 images dimensionnées en prenant le visage uniquement) et le modèle du papier de recherche (34023 images) :

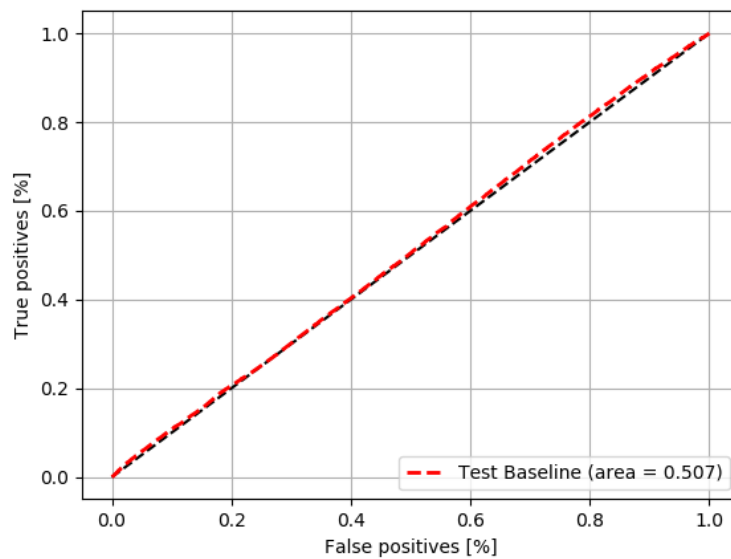


Figure 27 – Courbe ROC du modèle sur le jeu de données VIDTIMIT/DeepFakeTIMIT HQ

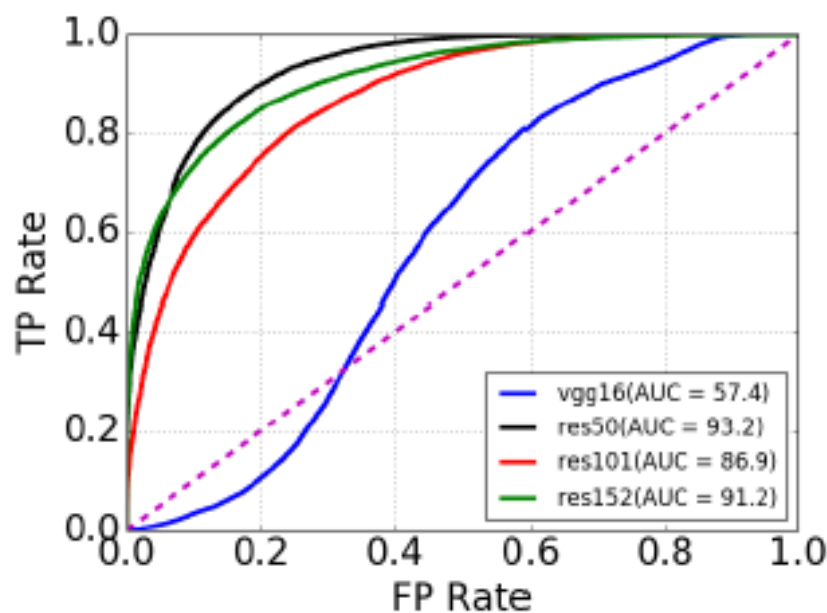


Figure 28 – Courbe ROC du modèle issu du papier de recherche sur le jeu de données DeepFakeTIMIT HQ



Le résultat de la performance sur DeepFakeTIMIT HQ est médiocre avec 50.7% comparé à 93,2% du papier de recherche. Cela est potentiellement dû au fait que la qualité ainsi que le processus de création de ces "DeepFake" (faceswap-GAN qui est le résultat d'un auto-encodeur dé-bruiteur avec GAN) soit trop différent de ceux en entraînement du modèle.

- Comparaison du test sur le jeu de données VidTIMIT/DeepFakeTIMIT LQ avec le modèle entraîné (27350 images dimensionnées en prenant le visage uniquement) et le modèle du papier de recherche (34023 images, l'intervalle d'extraction est certainement différent) :

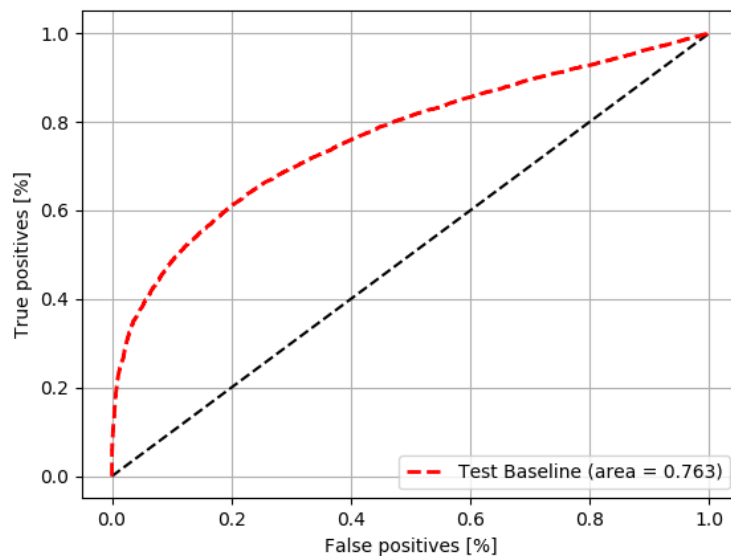


Figure 29 – Courbe ROC du modèle sur le jeu de données DeepFakeTIMIT LQ

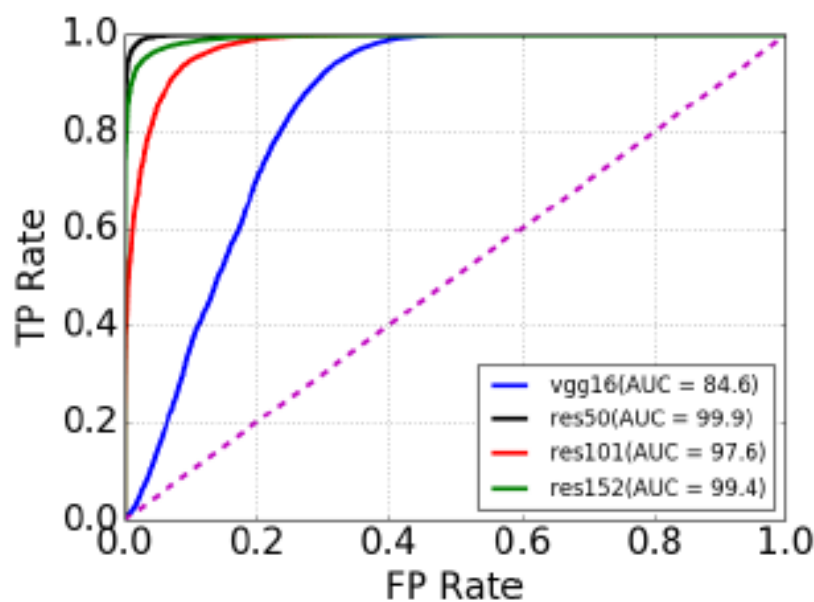


Figure 30 – Courbe ROC du modèle issu du papier de recherche sur le jeu de données DeepFakeTIMIT LQ

Le score AUC est bien meilleur lorsque la qualité est de faible résolution avec 76.3% de prédiction correcte, ce qui en fait la meilleure prédiction sur les bases du papier de recherche néanmoins très en deçà du score théorique de 99.9%.

Ces scores AUC décevants comparés à ceux du papier de recherche viennent très certainement d'une part du protocole de hard-mining qui n'a pas été effectué permettant certainement de limiter le nombre de faux positif lors de la prédiction et d'autre part de leur processus d'entraînement qui utilise uniquement des visages uniques rendant alors certainement la généralisation du modèle plus robuste qu'en utilisant un jeu de données avec seulement 1000 visages pour 1000 vidéos. La conclusion vis-à-vis du papier de recherche est que la performance du modèle entraîné n'est pas concluante par rapport à la capacité de prédiction attendu sur d'autres jeux de données que ceux des "DeepFake" de première génération.

Afin d'aller plus loin dans les tests, d'autres jeux de données sont disponibles dont Facebook Detection Challenge et FaceForensics++ (dont une partie est utilisée pour l'entraînement du modèle). Il est alors intéressant de voir la performance du modèle sur ceux non utilisés à savoir "DeepFake Detection" (Issu de plusieurs technologies dont FaceSwap, Face2Face, ...) composé de 1000 vidéos "DeepFake" issues de 363 vidéos d'acteurs et "NeuralTextures" [7] (procédé utilisant un auto-encodeur combiné à un GAN) composé de 1000 vidéos "DeepFake" issues de 1000 vidéos Youtube (identique au jeu de données DeepFakes) et Facebook Detection Challenge.

— Test sur le jeu de données DeepFake Detection :

Ce test est effectué sur l'ensemble des vidéos du jeu de données soit 23183 (50% positives / 50% négatives) images dimensionnées en prenant le visage uniquement.

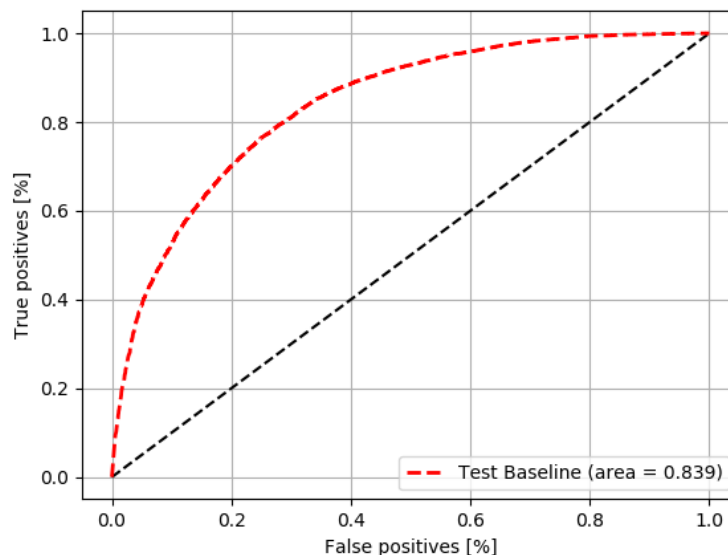
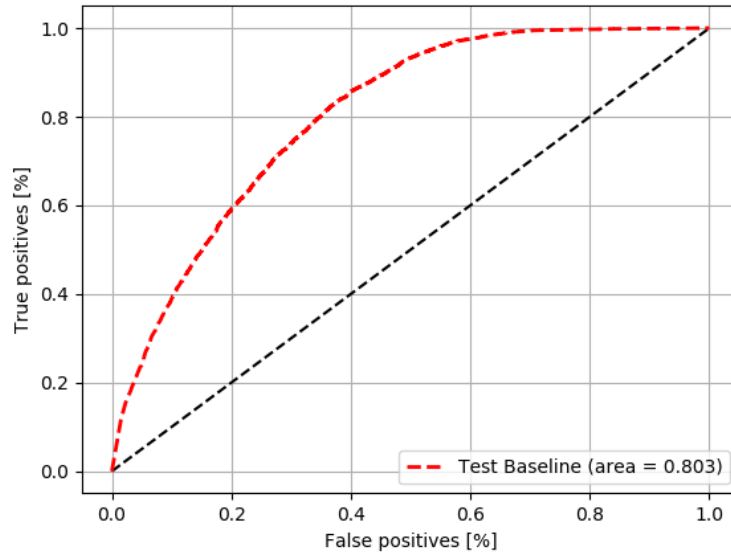


Figure 31 – Courbe ROC du modèle entraîné sur le jeu de données DeepFake Detection

Le score AUC réalisé sur ce jeu de données est de 83.9%, soit le meilleur sur l'ensemble des jeux de test. Cela démontre que malgré les scores précédents, l'entraînement avec le jeu de données a été pertinent sur la classification utilisant les technologies de première génération ciblés comme FaceSwap, Face2Face... qui sont les logiciels les plus communément utilisés.

— Test sur le jeu de données NeuralTextures :

Ce test est effectué sur l'ensemble des vidéos du jeu de données soit 31470 (50% positives / 50% négatives) images dimensionnées en prenant le visage uniquement.

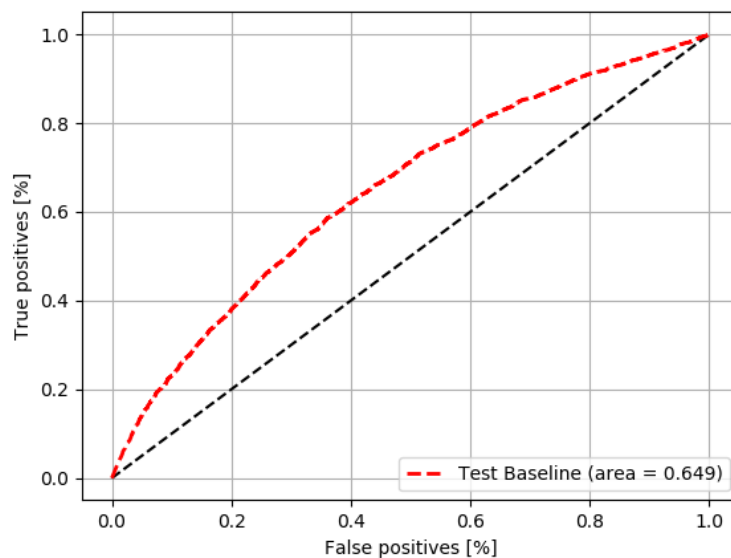


**Figure 32** – Courbe ROC du modèle entraîné sur le jeu de données NeuralTextures

Le score AUC est de 80,3%, ce qui est très correcte sachant qu'il s'agit d'une technologie qui utilise un GAN et un auto-encodeur, néanmoins ce score est à relativiser du fait que les images falsifiées et non falsifiées proviennent du même jeu de données que pour l'entraînement, ce qui rend donc l'analyse de ce score biaisée.

— Test sur le jeu de données DeepFake Detection Challenge :

Ce test est effectué sur un échantillon vidéo du jeu de données soit 12143 images (6408 positives / 5735 négatives) dimensionnées en prenant le visage uniquement.



**Figure 33** – Courbe ROC du modèle entraîné sur le jeu de données NeuralTextures

Le score AUC réalisé est de 64,9%, ce qui est plutôt correcte pour un tel modèle sachant que les "DeepFake" issues de ce jeu de données sont hétérogènes autant dans les technologies utilisées que des procédés (la voix peut être un exemple de seule source de "DeepFake" dans les données, ce qui peut être considérée comme un faux positif avec le modèle).

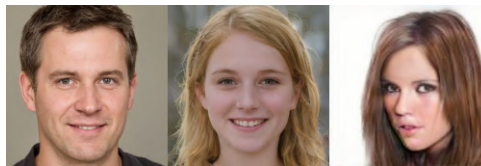
Les tableaux ci-dessous permettent de visualiser les scores AUC sur chacun des datasets testés :

Modèle	UADFV	DeepFakeTIMIT HQ	DeepFakeTIMIT LQ
ResNet50, $\eta = 0.001$ , fine-tune	0.601	0.501	0.763

Modèle	DeepFakeDetection	NeuralTextures
ResNet50, $\eta = 0.001$ , fine-tune	<b>0.839</b>	0.803

## 8 Protocole d'entraînement du réseau de neurones CNN

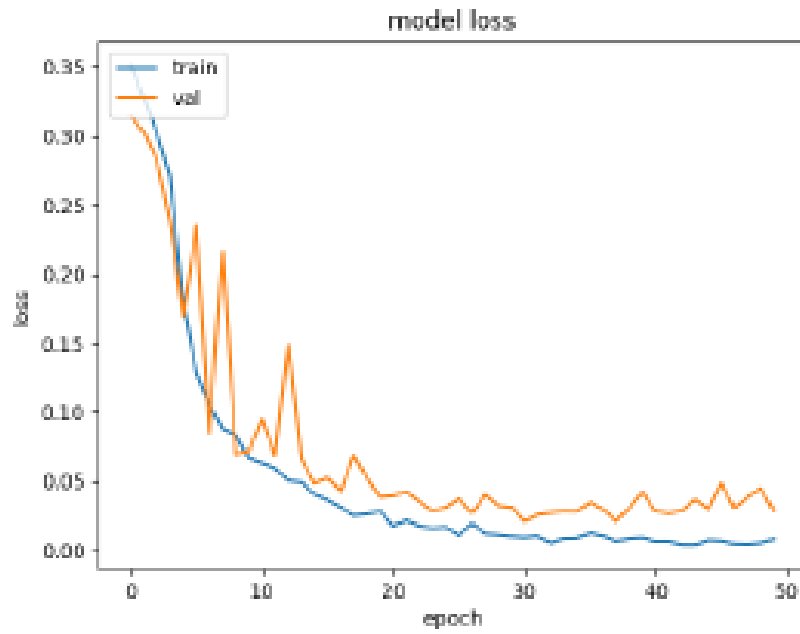
Cette section résume les différents entraînements, validations et tests du modèle CNN avec les différents essais effectués en utilisant le jeu de données Flickr et styleGAN2 fourni par NVIDIA. Un échantillon ci-dessous permet de comparer les différentes qualités de "DeepFake" issues des jeux de données de l'étude :



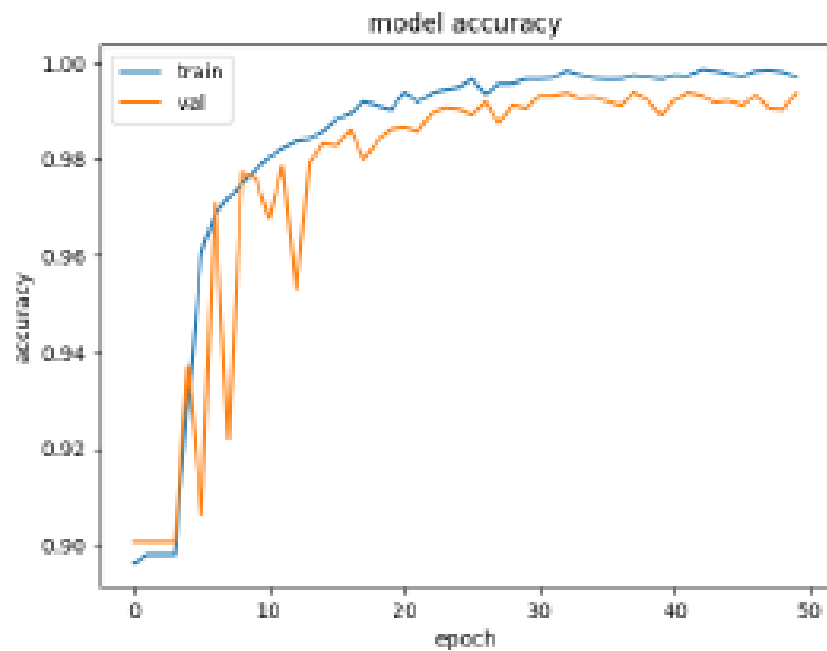
**Figure 34** – Échantillon d'images de gauche à droite venant de styleGAN2, styleGAN, StarGAN

Le jeu de données est composé de 30000 images positives dont sont issues les 30000 images "DeepFake" créés par styleGAN2. Les images ne subissent aucun traitement particulier sauf celui d'extraire leurs différentes matrices de co-occurrence (R,G,B).

A noter qu'il n'est pas mentionné explicitement le nombre et les degrés des matrices de co-occurrence extraites pour chaque image dans l'entraînement du modèle issu du papier de recherche. Il faut donc faire un choix arbitraire sur ces paramètres dans l'entraînement du modèle de l'étude. Les deux premiers essais sont réalisés en utilisant un unique degré ( $0^\circ$ ) des matrices de co-occurrence RGB alors que le dernier essai est réalisé avec l'ensemble des degrés des matrices de co-occurrence RGB dont l'injection est faite de manière aléatoire dans le but de constater un quelconque changement comparé aux autres essais. Les 4 matrices de degrés différents ont été produites avec la relation du plus proche voisin à droite, néanmoins l'étude pourrait être encore plus largement étendue par des matrices créées sur des relations différentes entre pixels. La première étape est de répartir ce jeu de données en 3 répertoires distincts ; entraînement, validation et test. La répartition du jeu de données étant mentionnée dans le papier de recherche, une séparation 50% entraînement, 25% validation et 25% test est donc réalisée. Le modèle est entraîné sur 50 epochs avec une taille de batch de 40. Avec la configuration utilisée pour l'entraînement, chaque epoch prend  $\pm 1$  heure et 20 minutes avec  $\pm 15$  minutes pour la validation et sauvegarde du modèle entre les epochs. Le modèle ne sera uniquement sauvegardé que dans le cas où sa précision augmente sur le jeu de données de validation par rapport à l'epoch précédente. Le papier de recherche fourni la courbe d'entraînement et validation (ci-dessous) de leur modèle qu'il sera alors possible de comparer à ceux du modèle entraîné sur le jeu de données mentionné précédemment.



**Figure 35** – Courbe de la perte (loss) du modèle issu du papier de recherche sur la base d'entraînement et validation



**Figure 36** – Courbe de la précision (accuracy) du modèle issu du papier de recherche sur la base d'entraînement et validation

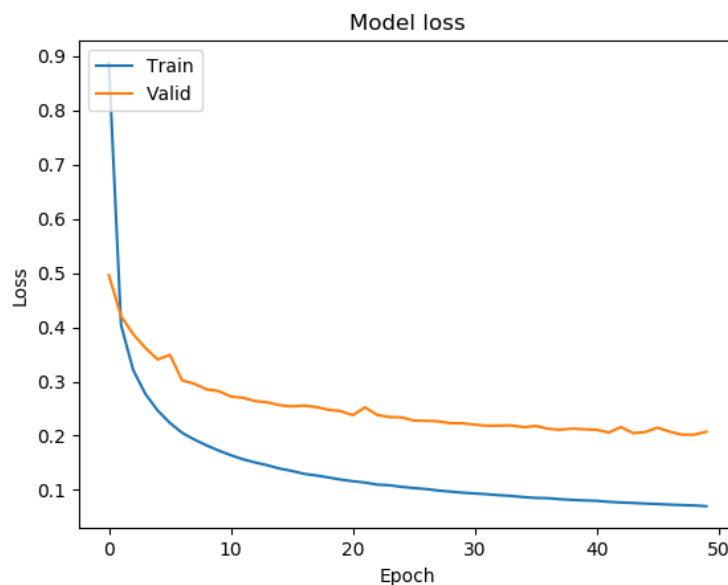
Essai n°1 **L'essai est réalisé avec le modèle utilisant l'optimiseur AdaGrad,  $\eta = 0.0001$ .**

La taille du jeu de données est de 60000 images (50% positives et 50% négatives). L'entraînement est donc composé de 30000 images, la validation est composée de 15000 images ainsi que pour le test (matrice de co-occurrence de degré 0). Les jeux de données sont mélangés 1000 fois pour obtenir une alternance aléatoire entre les deux classes.

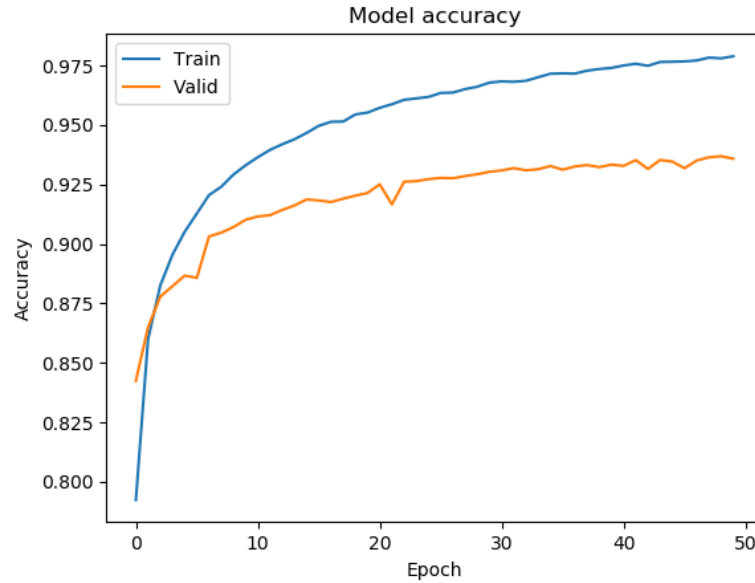
L'entraînement prend environ 1 heure et 20 minutes  $\times$  50 epochs soit environ 66 heures et 40 minutes. L'arrêt en avance (early stopping) est fixé à 50 epochs.

Le taux de  $\eta = 0.0001$  a été choisi du fait d'avoir à la première epoch des valeurs de perte (loss) très grande (37207.1479) lorsque la valeur de  $\eta = 0.0001$  donne une perte raisonnable à l'initialisation de 10.0441.

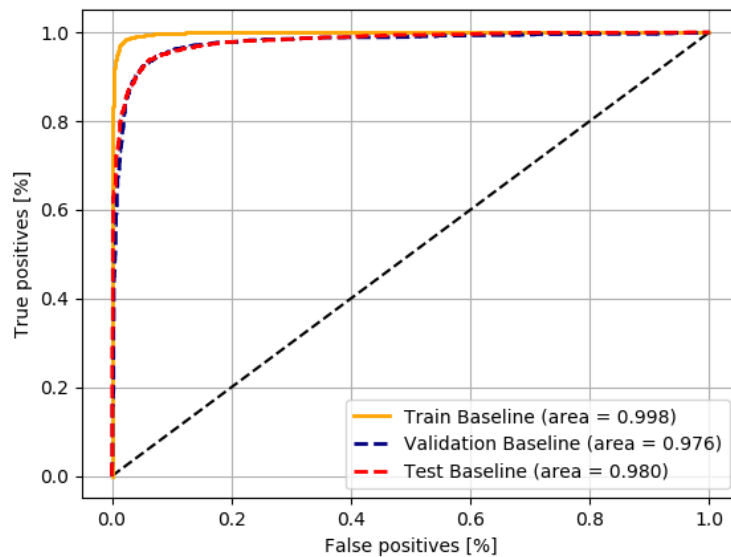
Au terme de l'entraînement, les résultats obtenus sont les suivants :



**Figure 37** – Courbe de perte (loss) du modèle CNN entraîné



**Figure 38** – Courbe de précision (accuracy) du modèle CNN entraîné



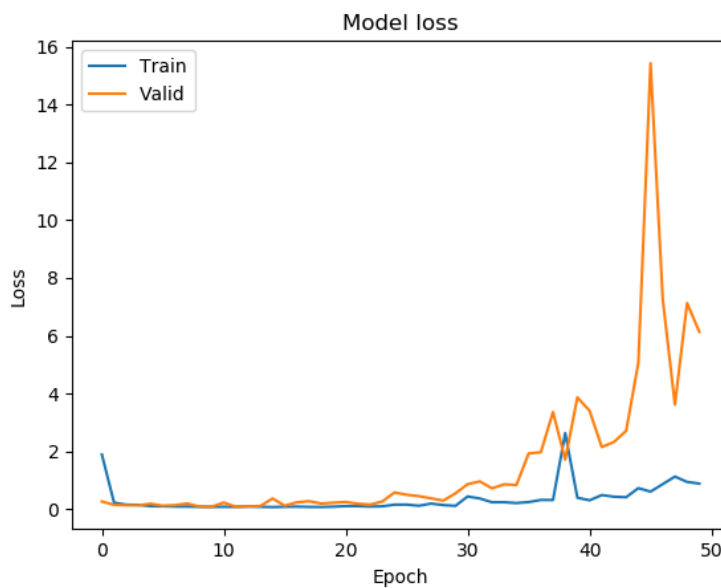
**Figure 39** – Courbe ROC du modèle CNN entraîné

Les courbes de perte (loss) indiquent que le modèle apprend correctement au vue de la courbe de perte des deux bases d'entraînement et de validation qui atteignent respectivement 0.090 et 0.1987.

Les courbes de précision (accuracy) montrent que la précision sur la base d'entraînement est de 0.9750 et de 0.9362 sur la base de validation. La perte est de 0.1987 et la précision de 0.9362 sur la base de test. Ces résultats démontre l'efficacité de méthode avec la matrice de co-occurrence qui est donc un excellent moyen de discrimination entre les visages non falsifiés et les "DeepFake" créés par GAN. Cela montre également que c'est adaptable à d'autres types d'architecture GAN que celle de StarGAN utilisée initialement. De plus, il suffit finalement d'une unique matrice de co-occurrence pour avoir une performance dans la classification supérieure à 90%. Néanmoins au vue des différences de marge entre les deux courbes la précision semble pouvoir encore tendre à être supérieure. Dans le

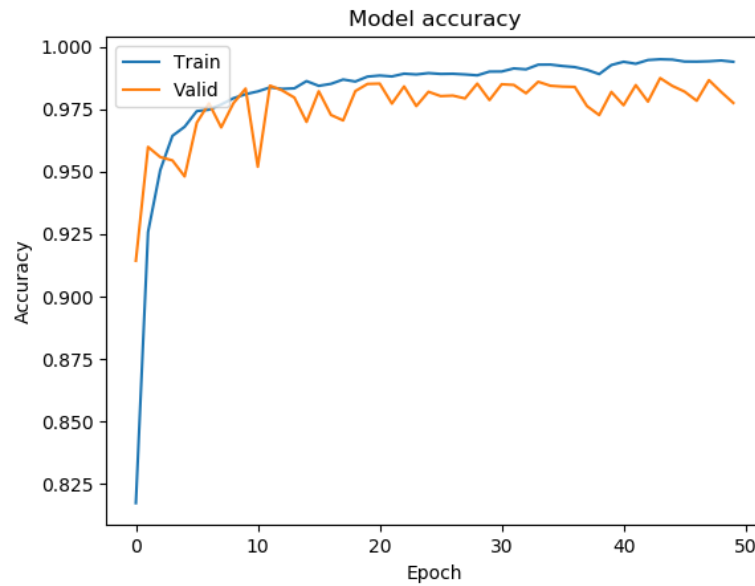
prochain essai, l'utilisation de AdaDelta devrait corriger le défaut visible de stagnation de la perte de AdaGrad donnant alors de meilleurs résultats. Par ailleurs, la courbe ROC montre que la prédiction est robuste avec 97.5% et 98.0% respectivement sur la base de validation et test.

Essai n°2 **L'essai est réalisé avec le modèle utilisant l'optimiseur AdaDelta,  $\eta = 1.0$  et  $\rho = 0.95$ .** La taille du jeu de données est de 60000 images (50% positives et 50% négatives). L'entraînement est donc composé de 30000 images, la validation est composée de 15000 images comme pour le test (matrice de co-occurrence de degré 0). Les jeux de données sont mélangés 1000 fois pour obtenir une alternance aléatoire entre les deux classes. L'entraînement prend environ 1 heure et 20 minutes  $\times$  50 epochs soit environ 66 heures et 40 minutes. L'arrêt en avance (early stopping) est fixé à 50 epochs. Au terme de l'entraînement, les résultats obtenus sont les suivants :

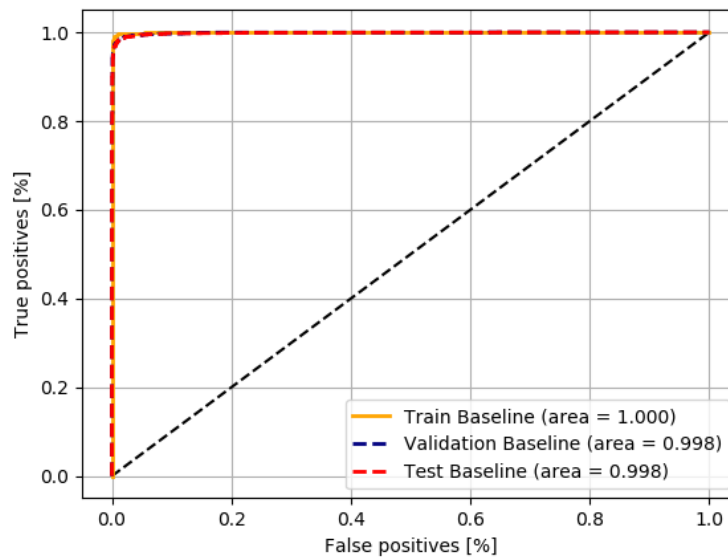


**Figure 40** – Courbe de perte (loss) du modèle CNN entraîné





**Figure 41** – Courbe de précision (accuracy) du modèle CNN entraîné



**Figure 42** – Courbe ROC du modèle CNN entraîné

Les courbes de perte (loss) montrent que le modèle apprend correctement avec néanmoins d'importantes oscillations par rapport à l'essai précédent. AdaDelta permet de diminuer encore plus la perte et d'augmenter la précision sur la base de validation et d'entraînement avec de meilleurs résultats qui sont largement atteints avant la fin des epochs (vers 14 epochs). Le minimum atteint en perte sur la base d'entraînement est de 0.0659 et sur la base de validation de 0.0756. Les courbes de précision (accuracy) montrent que le modèle possède une précision plus importante que l'essai précédent sur la base d'entraînement et sur la base de validation. Le maximum atteint en précision sur la base d'entraînement est de 0.9939 et sur la base de validation de 0.9775. La perte et précision sur la base de test sont respectivement de 5.7764 et 0.9785, ce qui est encore une fois meilleure qu'avec l'essai précédent sauf pour la perte qui est dû au fait qu'à la fin des epochs, la perte est largement remontée. Le changement d'optimiseur est donc une réussite dans

l'amélioration des résultats. Par ailleurs, la courbe ROC est légèrement meilleure que celle de l'essai précédent avec 99,8% sur la base de validation et 99,8% sur la base de test. En comparaison avec les courbes de perte et de précision du papier de recherche, les courbes de l'étude semblent tendre vers des valeurs équivalentes en perte et précision avec néanmoins une continuité dans la progression jusqu'à 50 epochs pour leurs courbes contrairement à celles de l'étude qui stagnent largement après une dizaine d'epochs. Ainsi, pour le prochain essai, le nombre d'époch avant arrêt (early stopping) passera de 50 à 30 epochs afin d'éviter les calculs inutiles au vue de la tendance des courbes.

Essai n°3 **L'essai est réalisé avec le modèle utilisant l'optimiseur AdaDelta,  $\eta = 1.0$  et  $\rho = 0.95$ .**

La taille du jeu de données est de 60000 images (50% positives et 50% négatives). L'entraînement est donc composé de 30000 images, la validation est composée de 15000 images comme pour le test (ensemble des matrices de co-occurrence). Les jeux de données sont mélangés 1000 fois pour obtenir une alternance nécessaire entre les deux classes.

L'entraînement prend environ 1 heure et 20 minutes  $\times$  50 epochs soit environ 66 heures et 40 minutes. L'arrêt en avance (early stopping) est fixé à 30 epochs.

Au terme de l'entraînement, les résultats obtenus sont les suivants :

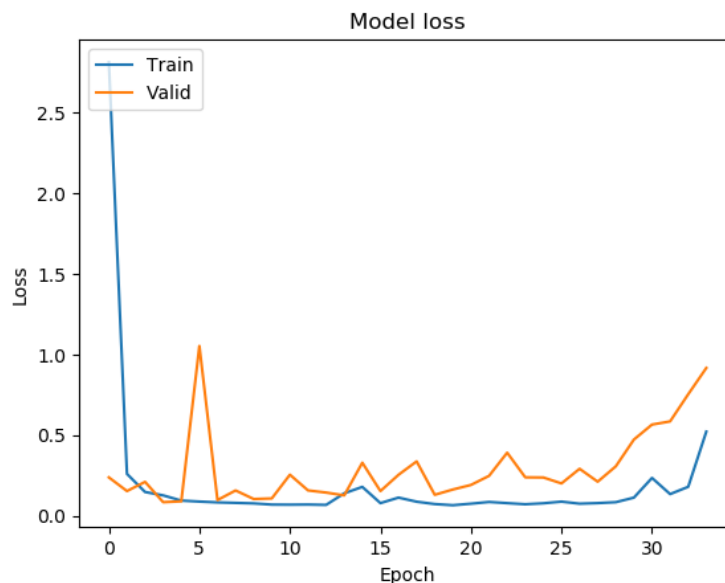


Figure 43 – Courbe de perte (loss) du modèle CNN entraîné

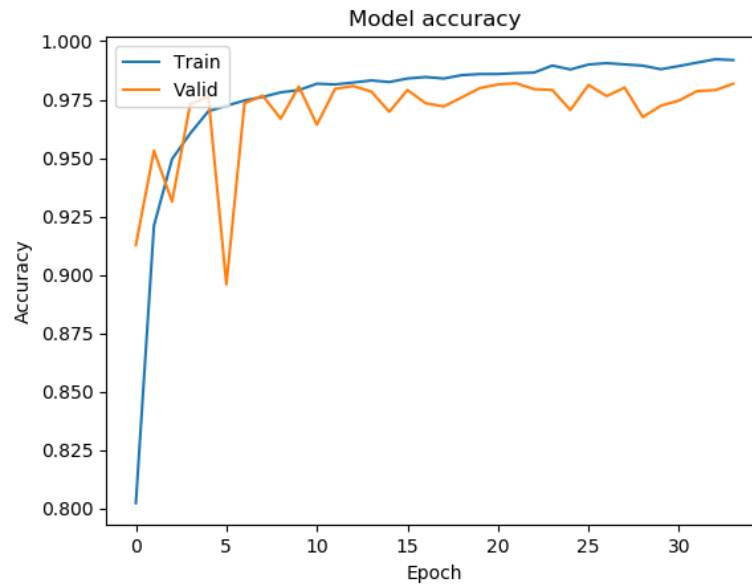


Figure 44 – Courbe de précision (accuracy) du modèle CNN entraîné

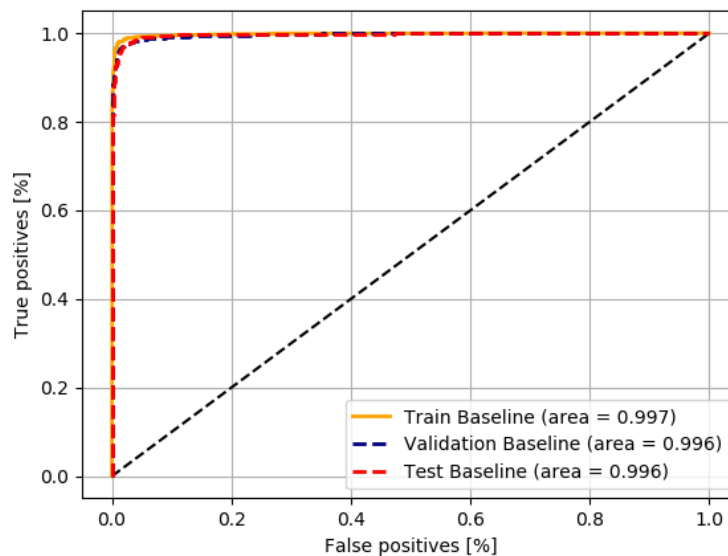


Figure 45 – Courbe ROC du modèle CNN entraîné

Les courbes de perte (loss) et de précisions montrent que le fait de faire passer aléatoirement un degré spécifique des matrices de co-occurrence ne permet pas une amélioration des résultats par rapport à l'utilisation d'une matrice de degré spécifique. La perte vaut à son minimum 0.0669 sur la base d'entraînement et 0.0857 sur la base de validation. La précision maximum est de 0.9919 sur la base d'entraînement et 0.9818 sur la base de validation, ce qui est inférieur de 0.0056 par rapport à l'essai précédent.

La perte sur la base de test est de 0.0833 et la précision de 0.9742 qui est inférieur de 0.0043 par rapport à l'essai précédent. Le fait de s'arrêter après une trentaine d'époques confirme que l'entraînement du réseau de neurones atteint bien sa performance maximum en avance par rapport aux 50 epochs initiaux. Les courbes ROC sont équivalentes au

dixième près au niveau de la performance par rapport à l'essai précédent.

Les tableaux ci-dessous permettent de résumer les différentes performances des essais :

Méthodes (entraînement)	Perte	Précision
CNN avec AdaGrad $\eta = 0.0001$ (un degré)	0.1987	0.9750
CNN avec AdaDelta $\eta = 1.0$ (un degré) et $\rho = 0.95$	<b>0.0659</b>	<b>0.9939</b>
CNN avec AdaDelta $\eta = 1.0$ et $\rho = 0.95$ (tout les degrés)	0.0669	0.9919
Méthodes (validation)	Perte	Précision
CNN avec AdaGrad $\eta = 0.0001$ (un degré)	0.090	0.9362
CNN avec AdaDelta $\eta = 1.0$ (un degré) et $\rho = 0.95$	<b>0.0756</b>	<b>0.9874</b>
CNN avec AdaDelta $\eta = 1.0$ et $\rho = 0.95$ (tout les degrés)	0.0857	0.9818
Méthodes (test)	Perte	Précision
CNN avec AdaGrad $\eta = 0.0001$ (un degré)	0.1987	0.9362
CNN avec AdaDelta $\eta = 1.0$ (un degré) et $\rho = 0.95$	5.7764	<b>0.9785</b>
CNN avec AdaDelta $\eta = 1.0$ et $\rho = 0.95$ (tout les degrés)	<b>0.0833</b>	0.9742

Sachant que le modèle de l'essai 2 et 3 sont très proche en performance, **le modèle sélectionné est l'essai 3** car il intègre dans son entraînement l'ensemble des matrices de co-occurrence ce qui le rend plus robuste sur les possibilités de degrés matricielle.

## 9 Protocole de tests du réseau de neurones CNN après entraînement

Dans cette section seront décrits les divers tests du modèle sélectionné précédemment.

- Comparaison du test sur le jeu de données StarGAN avec le modèle entraîné (23 888 images dont 50% en positives et 50% en négatives) et le modèle du papier de recherche (19990 images dont 1999 positives et 17991 négatives) :

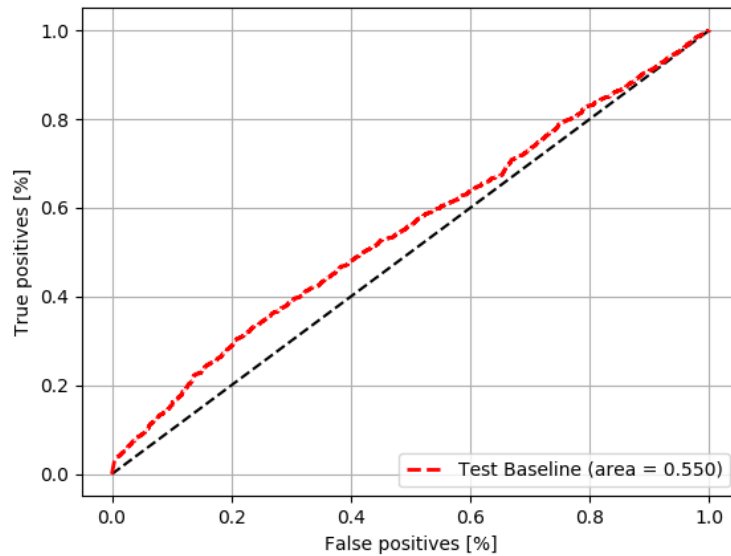
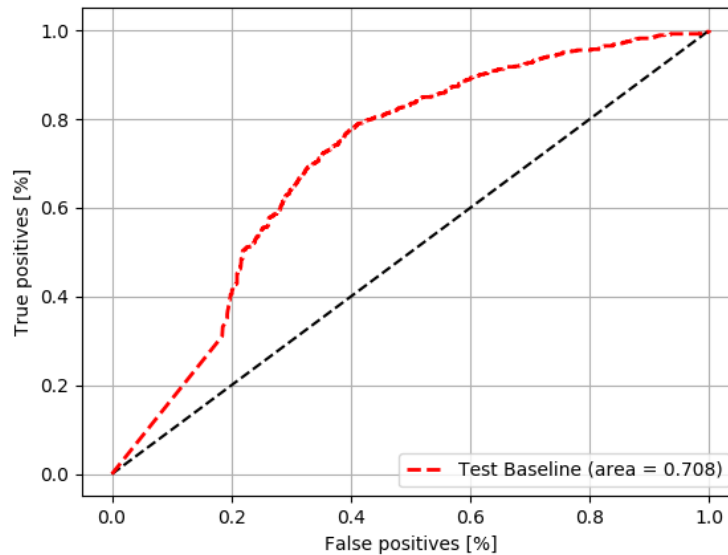


Figure 46 – Courbe ROC du modèle entraîné sur la base de test StarGAN

Le score AUC sur StarGAN est très en deçà de la performance du papier de recherche avec seulement 55,0% contre 99,8%, cela met en évidence que la matrice de co-occurrence est fortement liée à la technologie utilisée, StarGAN utilisant une technologie de génération différente de NVIDIA fait que la répartition statistique des pixels s'en trouve totalement changée. Il est néanmoins aussi possible que ce résultat puisse également être dû au fait que la différenciation soit située plutôt dans la multiplication des relations de pixels voisins que dans celle des degrés matricielle utilisée dans l'entraînement du modèle.

- Test sur le jeu de données StyleGAN (version 1) avec le modèle entraîné (2000 images dont 50% en positives et 50% en négatives) :



**Figure 47** – Courbe ROC du modèle entraîné sur la base de test StyleGAN

Le score AUC de 70,8% est bien meilleur que sur la base de StarGAN. Cela est certainement dû au fait qu'il s'agit de l'architecture de base du GAN de NVIDIA qui doit encore posséder des reliquats d'implémentations communes entre les deux versions.

En comparaison avec les courbes de perte et de précision du papier de recherche, leur modèle fait mieux en utilisant pour l'entraînement cycleGAN avec 99,45% sur le jeu de données StarGAN et inversement avec 93,42% contre 70,8% sur StyleGAN (version 1) et 50,5% sur StarGAN avec le modèle sélectionné. L'explication est certainement dû à la conception utilisée pour la création des différents "DeepFake" qui est trop hétérogène pour généraliser. La conclusion vis-à-vis du papier de recherche est donc que la performance du modèle de l'étude n'est pas concluante sur la généralisation à d'autres "DeepFake" créés par des modèles de GAN différents contrairement à ce qui aurait pu être attendu par rapport aux conclusions de celui-ci. Ce modèle sera néanmoins celui retenu pour l'implémentation du logiciel de détections de "DeepFake" compte tenu de ses différentes performances.

## 10 Visualisation avec l'interface

Dans l'étude, l'interface sert principalement à la visualisation des artefacts de l'image extraite d'une couche du réseau de neurones, la validation du modèle ayant déjà été prouvée par les divers tests sur des bases de "DeepFake". Le cas échéant, cela permet aussi aux enseignant-chercheurs de lever un possible doute sur une falsification grâce à la classification et à la visualisation. De multiples visualisations sont possibles grâce à la librairie tf-explain (feature map, vanilla gradient, ...), la plus adaptée pour l'étude étant GRAD-CAM [2] qui permet de visualiser les zones discriminantes qui ont permis au réseau de prendre la décision sur la classification de l'image.

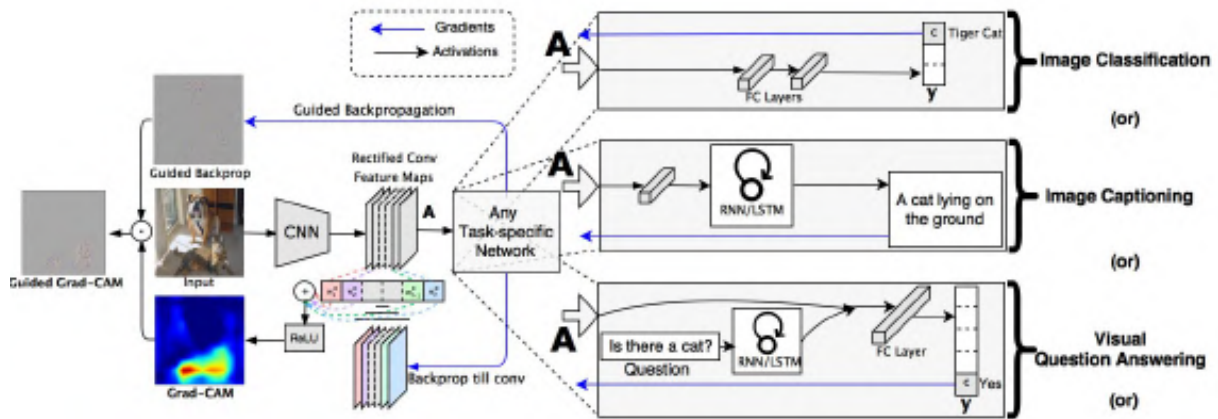


Figure 48 – Aperçu du fonctionnement de GRAD-CAM

Dans le cas de la classification d'images, les gradients de l'ensemble des classes sont mis à 0 sauf pour la classe de l'image qui sont mis à 1. Le signal est alors rétro-propagé dans les feature map d'intérêts qui sont combinés pour calculer les zones où le modèle regarde pour prendre une décision. En effet, les couches de neurones convolutionnelle retiennent les informations spatiales qui permettent la sémantique pour la classification et dont GRAD-CAM utilise les informations du gradient notamment dans la dernière couche de convolutions pour assigner les valeurs d'importances à chaque neurones par rapport à la zone d'intérêt de la décision.

Évidemment, chaque visualisation de couches d'un modèle est unique selon la configuration de son réseau ainsi que de l'image fourni. Pour la visualisation, le choix s'est porté sur le dernier bloc de convolution (conv5\_block3\_3\_conv) de ResNet50 car contenant les plus pertinentes informations de la segmentation dû à sa position dans le réseau. Pour le deuxième réseau de neurones, la dernière couche de convolution (conv2d\_5) est aussi choisi pour les raisons justifiées précédemment.

Le niveau de couleurs s'étend du bleu au violet dont l'échelle ci-dessous permet de pouvoir situer l'intensité du regard de la zone par le réseau de neurones.

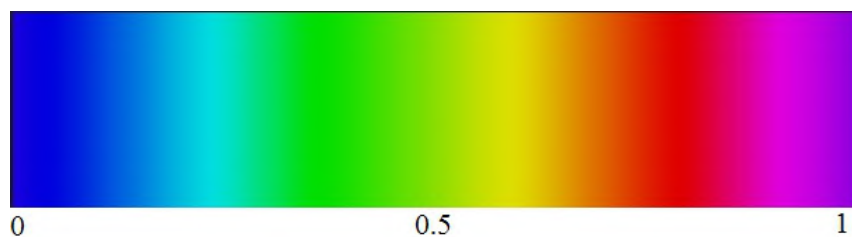
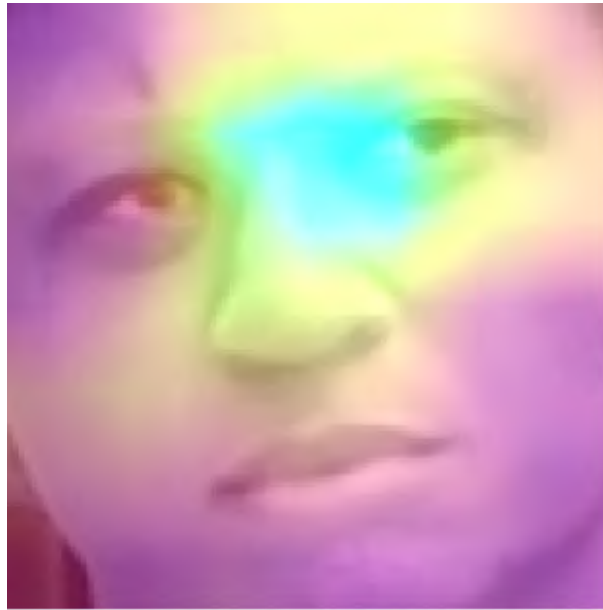


Figure 49 – Échelle de couleurs

Les exemples d'images ci-dessous permettent de comprendre la différenciation entre les modèles (détection d'images créées par auto-encodeur ou GAN).



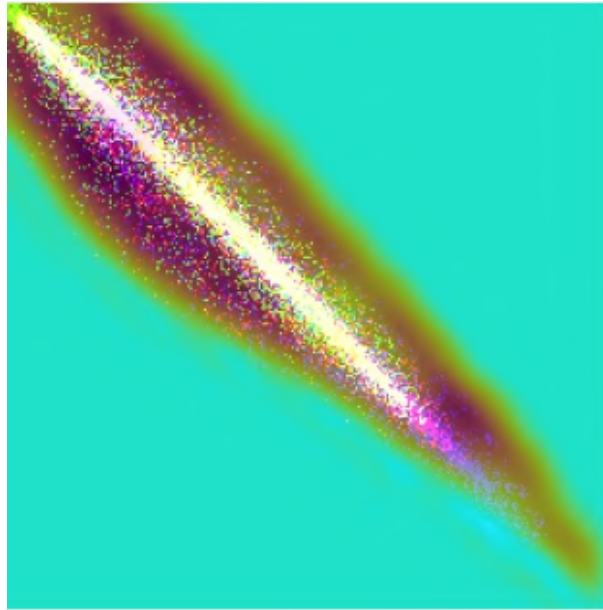
**Figure 50** – Zones d'intérêt dans la détection d'un DeepFake créé par auto-encodeur



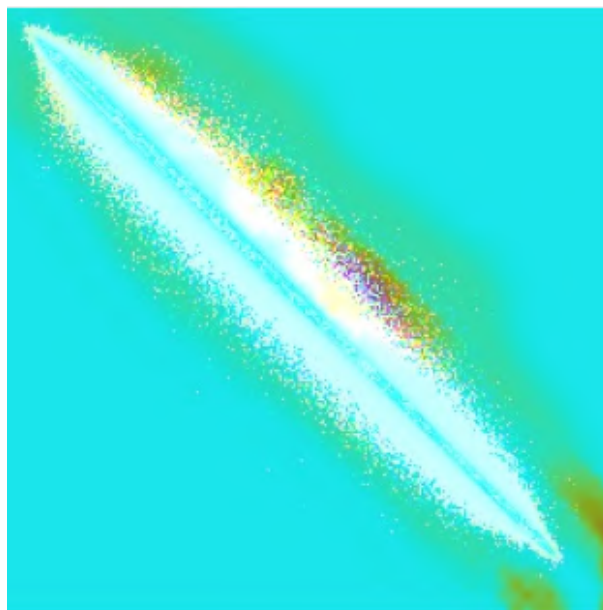
**Figure 51** – Zones d'intérêt de la détection d'un visage non falsifié par le détecteur de DeepFake créés par auto-encodeur

La visualisation entre les deux classes au niveau de la dernière couche de convolution de ResNet50 montre que les zones d'intérêt sont particulièrement situées en dehors de la zone des yeux. Il est probable que pour le réseau de neurones, ces emplacements soient les plus simples à interpréter pour faire une différenciation dans la falsification dû aux artefacts généralement présents lors de l'application du visage sur l'original. Par ailleurs, la zone des yeux est aussi regardée, néanmoins plus marginalement à cause de la possible difficulté dans le fait de détecter une falsification au niveau des yeux.





**Figure 52** – Zones d'intérêt de la détection d'un visage non falsifié par le détecteur de DeepFake créés par GAN



**Figure 53** – Zones d'intérêt dans la détection d'un DeepFake créé par GAN

La visualisation de la dernière couche est particulière avec le détecteur de "DeepFake" créés par GAN car les images en entrée ne sont constituées que par les matrices de co-occurrence RGB de l'image originale. Les zones d'intérêt sont alors uniquement sur la diagonale, les poids étant égaux à zéro ailleurs. Ainsi, la différence dans le regard du réseau de neurones entre les deux classes se fait principalement sur la périphérie de la diagonale avec une décision concentrée plutôt sur la partie haute (avec un pic d'intensité sur la partie basse de la diagonale) pour la détection d'un "DeepFake" contrairement à un visage non falsifié où le regard est alors plus diffus avec une zone plus large autour de la diagonale et très peu influente sur l'intérieur de celle-ci.

Ces visualisation prennent forme grâce à un logiciel qui permet de combiner l'ensemble des informations de la décision du réseau de neurones, à savoir l'étiquette, la confiance dans la prédiction et l'image associée à celle-ci.

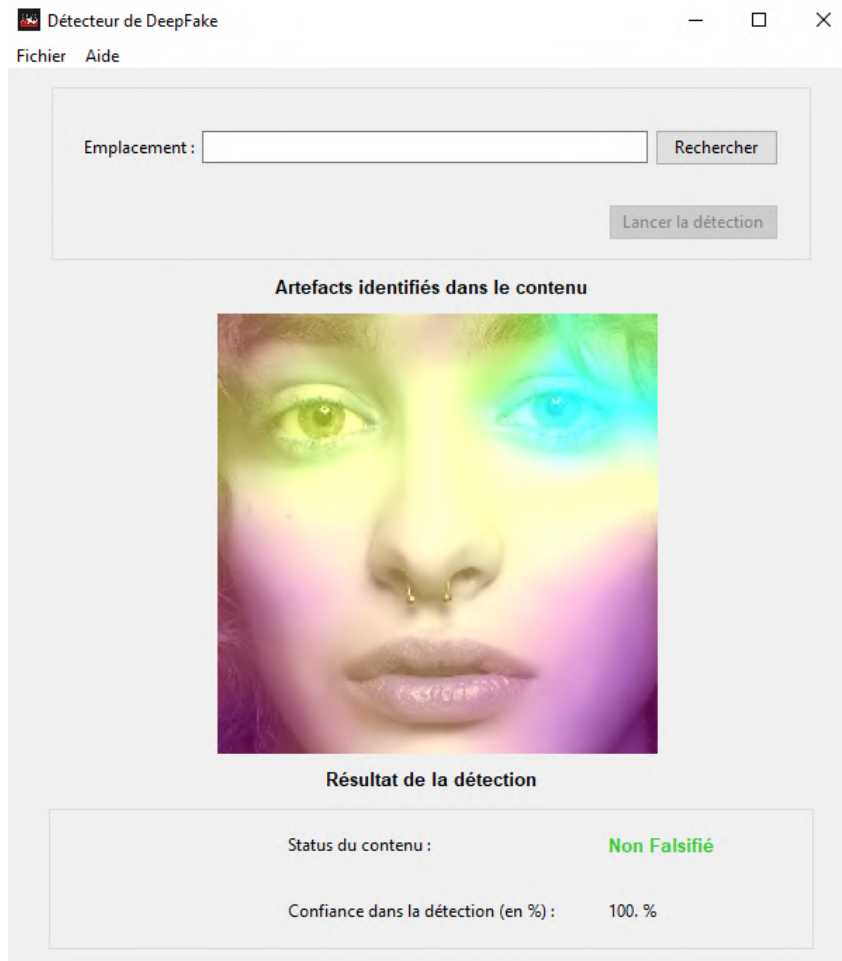


Figure 54 – Interface utilisateur

Au niveau de la prédiction qui doit s'afficher à l'utilisateur, la sortie des deux réseaux de neurones se fait selon la fonction d'activation sigmoïde, leur comparaison ne peut donc se faire que par la prédiction maximum selon la séparation des classes représentée par 0.5 et tel que :

$$\max(|(0.5 - prediction_{\text{détecteurAuto-encodeur}})|, |(0.5 - prediction_{\text{détecteurGAN}})|)$$

Lorsque l'entrée possède un format vidéo, ses frames sont extraites toutes les 1 secondes et une prédiction est réalisée sur chacune des images en retenant uniquement la prédiction pour laquelle la valeur est maximum avec  $\max(|(0.5 - prediction_{\text{Détecteur}})|)$ .

L'étiquette associée et les artefacts de l'image sont alors affichés en plus de la valeur de prédiction après comparaison entre les deux valeurs de prédictions des détecteurs selon la formule énoncée précédemment. La comparaison pose néanmoins le problème dans le cas où les deux prédictions sont égales (à N décimales près) et que leur étiquette sont opposées. La solution est donc d'indiquer à l'utilisateur que le logiciel n'a pas été en mesure de prendre une décision cohérente. Par ailleurs, un autre problème inhérent à la comparaison de modèles qui possèdent

un entraînement différent est la cohérence entre vrai-négatif et vrai-positif de chacun des modèles. En effet, un détecteur de "DeepFake" créés par auto-encodeur ne connaît pas la différence avec ceux générés par un GAN, ce qui peut alors mener vers une prédiction incorrecte (correcte au niveau de ses classes) supérieure à celle de l'autre modèle qui se trouve correcte pour le type d'images mais avec une confiance très légèrement inférieure. Ce phénomène peut donc parfois mener à avoir une visualisation et un étiquetage affiché à l'utilisateur incorrecte malgré la bonne analyse de celle-ci.

## 11 Analyse des résultats

Afin d'analyser au mieux les performances des modèles de deep learning, les métriques les plus pertinentes ont été utilisées tel que la précision qui permet de connaître sur chaque batch effectué par le modèle si celui-ci prédit de façon croissante les exemples étiquetés et dans le même temps que la perte (loss) diminue, ce qui permet d'indiquer que le modèle arrive à trouver un minimum local sinon global pertinent. (la loss doit être théoriquement la plus proche de zéro). L'évaluation du modèle se fait grâce à l'utilisation d'une base de validation à chaque fin d'époch. Les métriques sur cette base permettent de connaître si le modèle n'est pas en surapprentissage (overfitting) et qu'il arrive à généraliser les exemples avec une perte (loss) qui diminue et une précision qui augmente au cours des epochs. Enfin, une base de test permet de connaître après la fin de l'entraînement si celui-ci (i.e le réseau de neurones) possède les poids et biais qui permettent de généraliser sur une autre base indépendante. En plus de suivre les expérimentations par des courbes sur la perte et précision au cours des epochs, la métrique AUC (area under the curve) a aussi été utilisée afin de modéliser et comparer les courbes obtenues à celles des papiers de recherche qui ont permis les implémentations des modèles de l'étude et ainsi pouvoir formuler les conclusions adéquates.

La conclusion finale à retenir de l'étude est la difficulté pour un modèle d'avoir une capacité de performance constante proche des 100% dans l'ensemble des technologies que ce soit pour l'auto-encodeur ou GAN. En effet, les créations issues de ces technologies évoluent au fur et à mesure des recherches sur le deep learning (découverte du GAN en 2014) ce qui conduit à l'impossibilité de prédire les prochaines générations ou associations de modèles génératif.

Les "DeepFake" peuvent alors être assimilés à des virus par leur spécificité technologique de création. Ainsi, une méthode qui fonctionne aujourd'hui sur un ensemble de technologies n'est pas à l'abri de devenir obsolète sur celles de demain. Il s'agit donc d'une recherche permanente dont les différents challenges proposés par les Gafa tentent à leur façon d'accélérer.

## 12 Limites

Lors de l'évaluation d'un modèle de deep learning, la limite principale dans la recherche est finalement le temps car le deep learning résulte de la recherche par empirisme des meilleures hyperparamètres permettant d'avoir le meilleur résultat notamment dans la précision (accuracy) et la perte (loss) qui intervient dans la recherche du minimum global. Ainsi pour l'ensemble des modèles, il a été nécessaire de faire une vingtaine d'essais avec différentes architectures (nombre de neurones en sortie, de couches après le réseau pré-entraîné, choix du fine-tuning, de l'optimiseur, ...) et les différents paramètres associés (taux d'apprentissage, de décrétement, etc...). Une autre limitation est le fait de l'imprécision parfois importante dans la conception des réseaux de neurones énoncés dans les papiers de recherche. Enfin, l'étude est aussi conditionnée aux jeux de données disponibles au moment  $t$  sachant que de nouveaux datasets toujours plus importants en taille sortent périodiquement (2 à 3 mois).

## 13 Risques

Le risque associé au deep learning est principalement celui dû aux problèmes des jeux de données qui doivent être d'une certaine qualité, c'est à dire en nombre suffisant (généralement plusieurs dizaines de milliers d'images), être correctement étiquetés, avoir le format adéquat en entrée et surtout cibler suffisamment la zone d'intérêt pour que le réseau puisse déterminer correctement ce qui est d'intérêt pour une classification pertinente. Cela permet en plus des hyperparamètres de lutter contre des effets tels que le surapprentissage, ou le sous-apprentissage. Afin de prévenir ce risque, l'utilisation d'un arrêt en avance (early stopping) lors de l'entraînement permet d'arrêter l'apprentissage si le modèle n'arrive pas à converger vers une perte (loss) plus faible (intervalle de N epochs). Par ailleurs, la taille des bases de tests (validation et test) compte également et doit être suffisamment importante pour être significative dans l'évaluation du modèle. Le dernier risque à connaître qui est inhérent aux réseaux de neurones est celui d'être extrêmement spécialisé sur un type d'entrée car lié à la base qu'il doit apprendre, or les technologies disponibles pour créer des "DeepFake" sont hétérogènes, nécessitant d'avoir plusieurs modèles pour couvrir efficacement un plus large spectre de falsifications lorsque les méthodes utilisées ne parviennent pas à généraliser suffisamment sur d'autres types de technologies. Enfin, en cohérence avec l'analyse des risques initiale, un des modèles n'a pas pu être testé (CNN-LSTM) à cause d'un problème non identifié lors de la sauvegarde de celui-ci avec la librairie TensorFlow.

## 14 Outils et librairies

L'interpréteur utilisé est Conda (python 3.7), facilitant l'installation de certaines librairies comme Dlib. Les principales librairies utilisées sont Dlib (version 19.19.0) pour l'extraction des visages nécessaire au pré-traitement des images, Keras (version 2.3.1) pour l'implémentation des modèles de deep learning, TensorFlow, TensorFlow-estimator, TensorFlow-GPU, TensorFlow-GPU-estimator (version 2.1.0) nécessaires aux chargements des différents jeux de données et à l'utilisation du GPU pour les calculs ainsi que tf-explain (version 2.3.0) qui permet de visualiser le patch de convolution d'une couche de neurones particulière et enfin opencv-python (version 4.2.0.32) pour le pré-traitement des images, l'ouverture et l'enregistrement des images traitées.

## 15 Choix techniques

L'ensemble des entraînements, validations et tests pour chacun des modèles s'est fait sur le système d'exploitation Ubuntu 18.04.3 LTS 64 bits avec une machine possédant un processeur Intel Core i7-8700 cadencé à 3.20 Ghz sur 12 cœurs, 32 Go de RAM et une carte graphique NVIDIA RTX 2060 Super en GPU. L'utilisation de CUDA 10.0.0 ainsi que CuDNN 7.6.4 a été nécessaire afin de pouvoir effectuer les calculs sur GPU plutôt que CPU, beaucoup plus efficace en temps pour ce type de calcul par l'utilisation de la parallélisation.

# 6

## Bilan et conclusion

### 1 Bilan du semestre 9

#### 1.1 Tâches terminées

Le bilan à mi-parcours de ce projet de recherche et développement est positif, toutes les tâches du semestre 9 ont été réalisées, à date prévu et aucun retard n'est à signaler. A noter que le cahier de spécifications a dû subir des modifications induit par des ajouts de fonctionnalités ou de mises à jour syntaxique. Les tâches qui ont été terminées sont les suivantes :

- Rédaction et validation du cahier des charges
- Prise en main de certains logiciels de création de "DeepFake"
- Comprendre la structure des divers types de réseaux de neurones (CNN, LSTM, ...)
- Rédaction d'un manuel d'utilisation des logiciels de création de "DeepFake"
- Faire une revue de l'état de l'art sur la détection de "DeepFake"
- Délimiter les contours du sujet avec le tuteur / **MOA**
- Fixer les méthodes à implémenter pour la détection
- Rédaction d'une première puis deuxième version validée du cahier de spécifications
- Création de divers diagrammes du système (séquence, activité, classe, composant, ...)
- Création d'un diagramme de Gantt prévisionnel
- Création de la présentation du sujet de recherche et développement au moment du S9
- Rédaction d'une partie du rapport final correspondant au S9
- Prendre en main les diverses technologies de deep learning (TensorFlow, Keras, OpenCV, ...)
- Faire les comptes-rendus des diverses réunions suivis

#### 1.2 Tâches qui restent à faire

Les tâches qui restent à terminer sont les suivantes :

- Implémentation des divers classes défini par le diagramme de classes
- Création de la présentation du sujet de recherche et développement lors du S10
- Rédaction des parties restantes du rapport final correspondant au S10

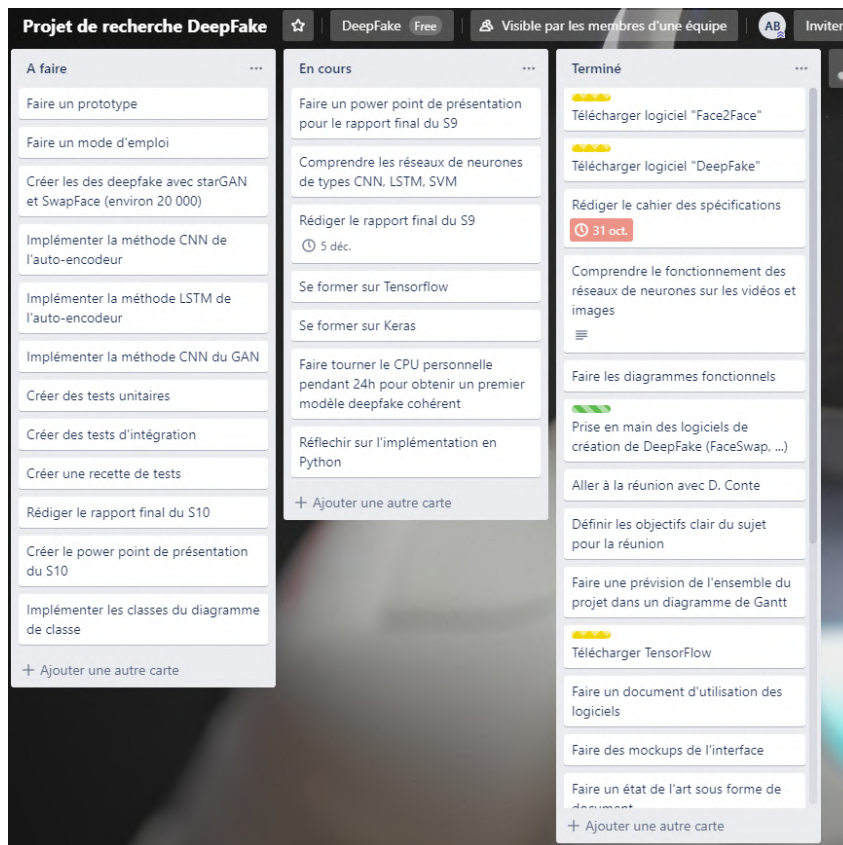


Figure 1 – Aperçu des tâches existantes grâce à un "Trello"

- Mettre à jour le cahier de spécifications
- Tester les modèles avec peu de données
- Tester les modèles avec beaucoup de données en utilisant StarGAN et FaceSwap
- Créer une recette de tests
- Créer des tests unitaires et d'intégrations
- Créer un mode d'emploi du logiciel
- Coder l'implémentation du système en python

## 2 Planning du semestre 10

Au niveau du diagramme de Gantt, un planning des tâches et notamment du développement à été réalisé. Ainsi, les phases de développements ont été découpées en jalons datés.

Un résumé des tâches primordiales à faire sont les suivantes :

- Conception de l'interface graphique
- Créer les jeux de données pour chaque modèle
- Implémenter les modèles de deep learning
- Tester les modèles de deep learning
- Rédaction des parties manquantes du rapport final
- Créer la présentation du projet de recherche et développement du S10
- Créer une recette de tests
- Créer des tests unitaires et d'intégrations
- Réfléchir sur l'implémentation en python

Une description plus exhaustive de l'ensemble des tâches est à retrouver dans la partie D, Gestion de projet de l'annexe.



### 3 Bilan du semestre 10

L'ensemble des tâches qui ont été programmées sur le S10 ont été réalisées ; à savoir la création des différents jeux de données, la rédaction des parties restantes du rapport final (Mise en œuvre, outils utilisés, limites, risques et bilans), la rédaction du cahier de tests et l'implémentation du logiciel avec les différents niveaux de tests (unitaire et intégration). L'ensemble des modèles retenus dans l'analyse sauf un (CNN-LSTM) ont été entièrement implémentés et testés avec leurs différents jeux de données.

### 4 Bilan sur la qualité

La qualité du code est assurée par plusieurs éléments :

- Tests unitaires
- Tests d'intégrations
- GitLab Continuous Integration (CI)
- Guide utilisateur
- Documentation développeur
- Documentation du code source (PyDoc)
- Cahier de tests

Cette mise en place d'outils permet de réduire la dette technique et donc de faciliter une possible reprise du logiciel ou maintenance de celui-ci.

### 5 Bilan auto-critique sur la gestion

Le premier semestre s'est bien déroulé dans l'ensemble, le planning a été tenu voire complété en avance d'une semaine. Une seule réunion a été nécessaire dû au fait que ce projet de recherche était initialement assez ouvert et que donc peu de contraintes venait du client, mes propositions étant en majorités celles du cahier des charges. L'état de l'art a été fait avec rigueur afin de pouvoir démarrer sur des modèles qui possèdent une précision d'au moins 90% théorique et dont l'implémentation ne requière pas une trop grande complexité. Lors du second semestre, l'implémentation du logiciel a été au cœur des échanges entre l'encadrant pédagogique et moi-même afin d'établir l'aspect attendu du logiciel et les évaluations à faire des modèles. L'attente du sujet dans son ensemble est conforme au niveau logiciel et de l'analyse car répondant aux critères de visualisations et d'analyses d'une gamme large de falsifications (auto-encodeur ou GAN) même si le modèle CNN-LSTM n'a pas pu être testé. Néanmoins, le principal inconvénient qui subsiste est que le score AUC des différents modèles de détections soit finalement assez médiocre sur certains jeux de tests (auto-encodeur et GAN) comparé à celui attendu vis-à-vis des papiers de recherche pouvant amener à une amélioration de ceux déjà existants ou par l'ajout de nouveaux modèles.

## Annexes



# A

## Description des interfaces externes du logiciel

### 1 Interface matériel/logiciel

Lors de l'utilisation du logiciel, le traitement par défaut de la vidéo ou de l'image se fait par utilisation du CPU, néanmoins il est plus que recommandé d'utiliser le GPU (compatible CUDA®) afin d'accélérer les traitements. En effet, l'architecture d'un GPU est optimisée pour traiter ce type d'entrée ainsi que l'utilisation des modèles de deep learning. Cela vient du fait que le GPU peut calculer en parallèle beaucoup plus de données par sa plus grande bande passante et nombre de cœurs comparé à un CPU.

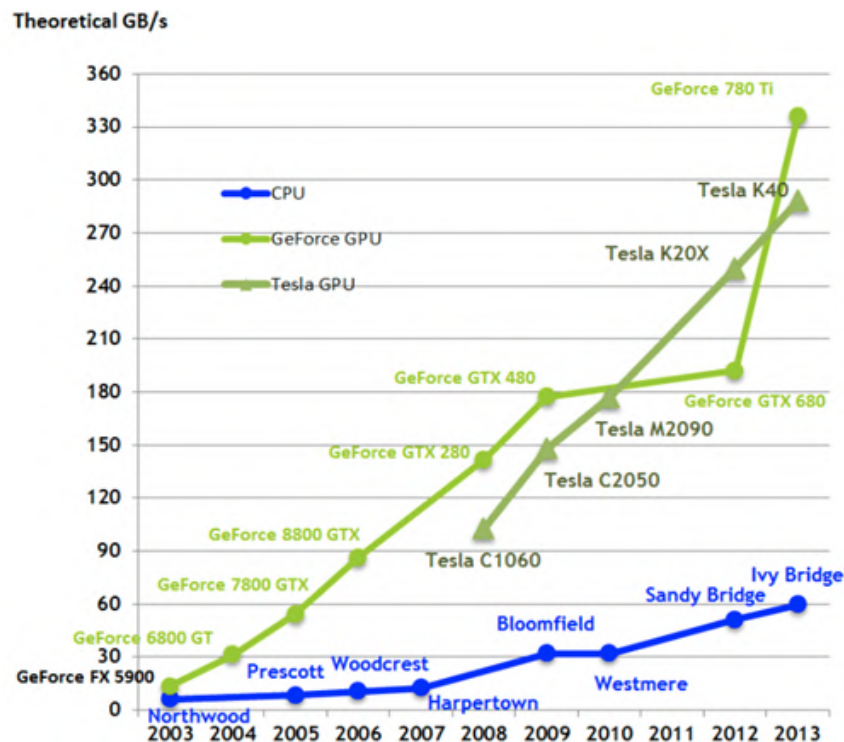


Figure 1 – Comparaison dans le temps (entre 2003 et 2013) de la bande passante entre CPU et GPU

## 2 Interface homme/machine

Le système sera composé d'une seule IHM simple avec une seule entrée pour l'utilisateur. L'identification du contenu utilisera plusieurs modèles pré-entraînés et où la seule tâche de l'utilisateur sera d'indiquer le dossier dans lequel se trouve le contenu à examiner. Les messages d'erreurs se feront par l'apparition de modale décrivant l'exception soulevée par l'utilisation. Afin de pouvoir orienter le design de l'application au type d'utilisateur potentiel, l'utilisation des principales lois de la Gestalt (proximité, clôture, familiarité) permettra d'avoir une interface ergonomique adéquate.

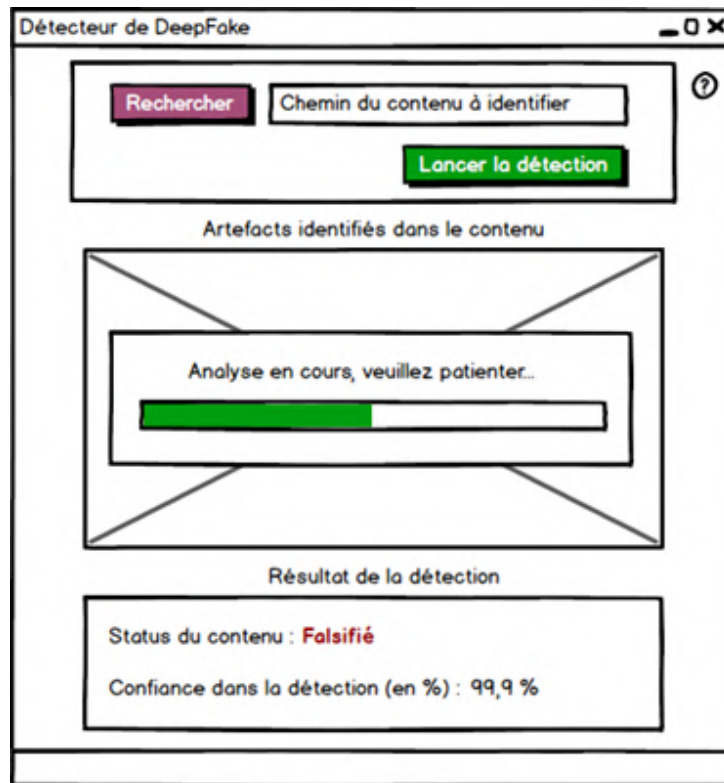


Figure 2 – Maquette de l'interface homme/machine

## 3 Interfaces logiciel/logiciel

Ce logiciel ne dépend d'aucun autre logiciel. L'entraînement des modèles d'apprentissage profond utilisera des données issues de plusieurs fichiers ou jeux de données contenant eux-mêmes des milliers d'images et vidéos afin de permettre à l'utilisateur de ne pas avoir à les entraîner pour les utiliser.

# B

## Spécifications fonctionnelles

### 1 Vue globale des fonctionnalités du programme de détection de "DeepFake"

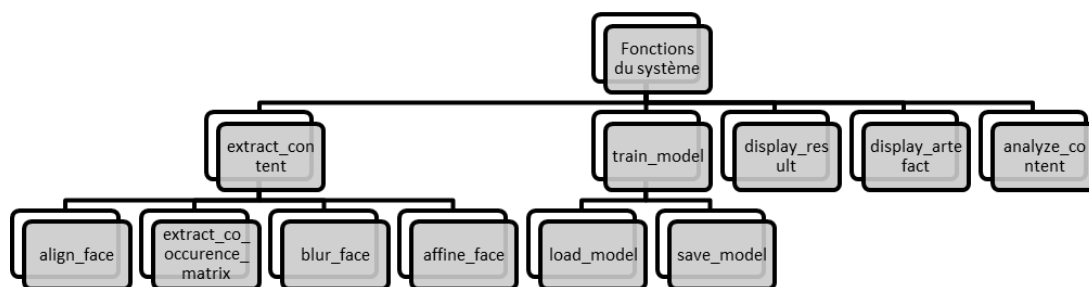


Figure 1 – Arbre hiérarchique des fonctionnalités

### 2 Définition de la fonction 1 : extract\_content

#### 2.1 Identification de la fonction 1

Cette fonction permet de pouvoir extraire le contenu de l'image ou de la vidéo (sous forme de multiples images) et de retourner une ou plusieurs autres images avec un cadrage centré sur le visage. Priorité : Primordiale.

#### 2.2 Description de la fonction 1

Cette fonction prend en entrée une image, un shape\_predictor et en sortie une image. La précondition est d'être une image ou vidéo avec un ou plusieurs individus. L'interface homme/machine, la fonction analyze\_content et train\_model sont les principales interactions avec cette fonction.

### 3 Définition de la fonction 2 : align\_face

#### 3.1 Identification de la fonction 2

Cette fonction permet de pouvoir aligner les différents visages de chaque image et de retourner une autre image avec un cadrage homogène au niveau de l'orientation et du zoom. Priorité : Primordiale.

#### 3.2 Description de la fonction 2

Cette fonction prend en entrée une image et en sortie une image. La précondition est d'être une image possédant un contenu avec visage. La fonction extract\_content interagit avec cette fonction.

### 4 Définition de la fonction 3 : blur\_face

#### 4.1 Identification de la fonction 3

Cette fonction permet d'appliquer une fonction gaussienne sur chaque visage afin de pouvoir faire ressortir plus facilement les artefacts de la fonction affine qui a pu être appliquée dessus. Priorité : Primordiale.

#### 4.2 Description de la fonction 3

Cette fonction prend en entrée une image et en sortie une image. La précondition est d'être une image possédant un contenu avec visage. La fonction align\_face interagit avec cette fonction.

$$g(i, j) = \sum_{k, l} f(i + k, j + l) h(k, l)$$

Figure 2 – Formule du filtre appliqué sur chaque pixel

### 5 Définition de la fonction 4 : affine\_face

#### 5.1 Identification de la fonction 4

Cette fonction permet d'appliquer une transformation affine sur le visage qui a subi un floutage et de le réappliquer sur le visage d'origine afin de simuler un « DeepFake » sans avoir recours à un traitement logiciel coûteux en temps. Priorité : Primordiale.

#### 5.2 Description de la fonction 4

Cette fonction prend en entrée une image et retourne en sortie une image. align\_face interagit avec cette fonction.

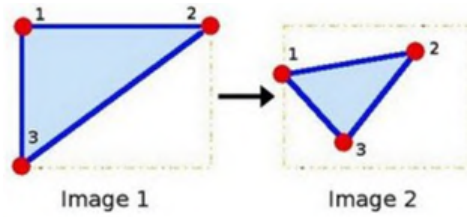


Figure 3 – Utilisation des points du contours visage pour aligner l'image sur l'autre

## 6 Définition de la fonction 5 : `extract_co_occurrence_matrix`

### 6.1 Identification de la fonction 5

Cette fonction permet d'extraire la matrice de co-occurrence de chaque couleur de l'image GAN qui sera ensuite utilisée par le modèle pour détecter s'il s'agit d'une manipulation d'image ou non. Priorité : Primordiale

### 6.2 Description de la fonction 5

Cette fonction prend en entrée une image et retourne en sortie une matrice. `train_model` interagit avec cette fonction. Priorité : Primordiale

$$C_{\Delta x, \Delta y}(i, j) = \sum_{x=1}^n \sum_{y=1}^m \begin{cases} 1, & \text{if } I(x, y) = i \text{ and } I(x + \Delta x, y + \Delta y) = j \\ 0, & \text{otherwise} \end{cases}$$

Figure 4 – Formule de la matrice de co-occurrence  $C$

## 7 Définition de la fonction 6 : `train_model`

### 7.1 Identification de la fonction 6

Cette fonction permet d'entraîner les divers modèles de deep learning choisis dans ce sujet afin de détecter les contenus « DeepFake ». Priorité : Primordiale

### 7.2 Description de la fonction 6

Cette fonction prend en entrée une liste d'images, des paramètres de configuration et retourne en sortie un modèle entraîné.

## 8 Définition de la fonction 7 : `analyze_content`

### 8.1 Identification de la fonction 7

Cette fonction permet de faire une analyse du contenu en entrée grâce aux différents modèles entraînés préalablement et d'obtenir à la suite, un indicateur numérique sur sa possible falsification. Priorité : Primordiale

### 8.2 Description de la fonction 7

Cette fonction prend en entrée une liste d'images, un modèle de deep learning entraîné et retourne en sortie un indicateur sous forme de valeur.

## 9 Définition de la fonction 8 : `display_result`

### 9.1 Identification de la fonction 8

Cette fonction permet d'afficher le résultat de l'analyse du contenu. Priorité : Primordiale

### 9.2 Description de la fonction 8

Cette fonction prend en entrée une valeur et ne retourne rien.

## 10 Définition de la fonction 9 : `display_artefact`

### 10.1 Identification de la fonction 9

Cette fonction permet d'afficher les artefacts de l'image qui ont permis au modèle de deep learning de pouvoir conclure sur le contenu. Priorité : Secondaire

### 10.2 Description de la fonction 9

Cette fonction prend en entrée une image et ne retourne rien.

## 11 Définition de la fonction 10 : `load_model`

### 11.1 Identification de la fonction 10

Cette fonction permet de charger un modèle pré-entraîné. Priorité : Secondaire

## 11.2 Description de la fonction 10

Cette fonction prend en entrée un modèle pré-entraîné et ne retourne rien.

## 12 Définition de la fonction 11 : save\_model

### 12.1 Identification de la fonction 11

Cette fonction permet de sauvegarder un modèle pré-entraîné sous forme de fichier. Priorité : Secondaire

### 12.2 Description de la fonction 11

Cette fonction ne prend rien en entrée et ne retourne rien.

# C

# Spécifications non fonctionnelles

## 1 Contraintes de développement et conception

### 1.1 Contrainte matériels

La principale contrainte dans ce projet est au niveau matériel. En effet, un entraînement de plusieurs dizaines de milliers d'entrées est nécessaire pour avoir un modèle de deep learning possédant une résolution fine. Cela amène donc à posséder une grande capacité de calcul au niveau CPU, GPU et de la RAM. Lors des premiers essais, un modèle pré-entraîné simple ou accessible sur Internet sera utilisé mais lors d'essais plus avancés sur une falsification de meilleure qualité, cela devra requérir la construction d'un modèle possédant un temps d'entraînement assez important (plusieurs heures). Selon la configuration minimum, un CPU de base Intel ou AMD est suffisant, néanmoins afin de raccourcir la durée du calcul, il est recommandé d'utiliser un GPU NVIDIA avec un minimum de 4 Go de RAM ainsi qu'une compatibilité CUDA® (score de capacité de traitement de 3.5 minimum) nécessaire avec TensorFlow-GPU.

### 1.2 Contrainte de langage de programmation

Lorsqu'il est question de traitement d'images et d'apprentissage profond, une multitude de langage peut être utilisé tel que Java, C++, Python, ... Néanmoins, Python disposant déjà de bibliothèques très nombreuses (TensorFlow, Keras, Pytorch, OpenCV, ...) et d'un large support communautaire dans ce domaine, le choix de son utilisation a été acté pour ce projet.

### 1.3 Contrainte logiciels

Le choix des bibliothèques Python utilisées a été porté sur l'implémentation des types de réseaux de neurones utilisés lors des études. L'implémentation des modèles se fera grâce aux bibliothèques TensorFlow, TensorFlow-GPU, Keras et OpenCV. La bibliothèque WxWidget sera utilisé pour l'interface graphique dû à sa grande portabilité sachant que le logiciel pourra être installé sur plusieurs types de supports différents en fonction des enseignants-chercheurs qui souhaitent l'utiliser. Afin de reproduire les conditions des papiers de recherche lors des



expérimentations des techniques de détection, les jeux de données issu de ceux-ci (VidTIMIT, HOHA, GRID, UADFV, ...) seront en priorité utilisés si possible. Il sera possible à la suite d'utiliser d'autres jeux de données avec les modèles d'apprentissage profond et pouvoir en tirer les conclusions adéquate.

## 1.4 Contrainte environnementale

L'environnement pour le projet utilisera l'IDE PyCharm (version 2019.2) de JetBrains avec l'interpréteur système de Python (version 3.7.4).

# 2 Contraintes de fonctionnement et d'exploitation

## 2.1 Performances

S'agissant d'un sujet exploratoire, son rôle est principalement la démonstration des méthodes de détection existant sur les contenus « DeepFake ». Ainsi les spécifications temps réel liées à l'utilisation du système du point de vue utilisateur et environnement ne seront pas impératif pour ce projet. Néanmoins, les modèles retenus seront uniquement ceux qui permettent d'avoir une confiance dans la détection de DeepFake à plus de 90% afin d'avoir une utilisation pertinente du logiciel.

## 2.2 Capacités

Le nombre de terminaux pour l'utilisation du logiciel n'est pas limité à un nombre, néanmoins son installation sera restreinte aux enseignants-chercheurs souhaitant l'utiliser, avec comme support un ordinateur détenant une carte graphique. La taille maximum des données traitées n'est pas limitée, mais son temps de traitement sera sujet à celui disponible par l'utilisateur. La capacité maximum de stockage sera assujettie à la machine utilisant le logiciel. Les transactions simultanées ne sont pas existantes sur ce système.

## 2.3 Modes de fonctionnement

En cas d'arrêt du système lors de son entraînement, le système sauvegarde automatiquement le modèle en construction et permet de reprendre sa construction ultérieurement. En cas de problème avec l'interface graphique, un mode dégradé sous forme de ligne de commande permettra d'utiliser le logiciel.

## 2.4 Contrôlabilité

Un affichage sera présent en mode ligne de commande sur la console afin de pouvoir suivre le déroulement de l'algorithme et d'éventuelle problème. En cas d'erreur, un fichier de log sera généré afin d'en préciser les causes.

## 2.5 Sécurité

Comme défini précédemment, l'utilisation du logiciel n'est pas restreinte par un accès sécurisé. La seule condition d'utilisation est l'utilisation en local qui ne sera disponible que pour les enseignants-chercheurs de l'école de journalisme de Tours.

## 2.6 Intégrité

La déconnexion imprévue du système lors de l'entraînement du modèle peut causer une perte du modèle ou sa corruption. La restauration du système n'existe pas dans ce cas et il faut donc le reconstruire depuis le début. Ceci est également vrai pour le processus d'analyse de l'entrée de l'utilisateur.

## 2.7 Maintenance et évolution du système

Le système reposant avant tout sur une validation d'un modèle de recherche par empirisme, l'ensemble des méthodes existantes ou futures n'ont pas été implémentés. Il est aussi possible d'avoir d'autres méthodes de falsification non abordées dans ce sujet. L'évolution du système pourra passer par l'ajout de nouvelles méthodes de détection avec l'aide d'autres types de réseaux de neurones implémentés ou sinon de méthodes dans la détection de formes (objet, ...) voire l'ajout d'une option pour l'utilisateur lui permettant de directement entraîner les modèles existants avec de nouveaux jeux de données.

# D


























## Gestion de projet

Comme défini dans le cahier des spécifications, la méthode utilisée pour ce projet est en Agilité. Cela permet de rendre compte de façon régulière sur les avancés du projet et notamment du livrable.

Les outils utilisés pour ce projet de recherche et développement sont :

- planification des tâches avec Microsoft Project
- gestion des tâches avec "Trello"
- compte-rendu hebdomadaire avec le tuteur pédagogique
- gestion des versions avec Git
- hébergement du code source et CI avec GitLab

Le développement du projet se fait en mode Agile avec des sprints régulier qui durent au maximum 3 semaines avec le développement des fonctionnalités en finissant par les tests unitaires et par ordre d'importance pour le client.

N°	 Mode Tâche	Nom de la tâche	Durée	Début	Fin	Sep 19	Oct 19	Nov 19	Déc 19	Jan 20	Fév 20	Mar 20	Avr 20			
						09/16/23	30/07/14	21/28/04	11/18/25	02/09/16	23/30/06	13/20/27	03/10/17	24/02/09	16/23/30	06/06/06
1			Prise en main des logiciels de DeepFake (Face2Face et FaceSwap)	9 jours	Mer 18/09/19	Lun 30/09/19										
2			Recherche des méthodes de détection existante sur le DeepFake	9 jours	Mer 18/09/19	Lun 30/09/19										
3			Réunion avec N.Ragot	1 jour	Lun 23/09/19	Lun 23/09/19										
4			Rédaction des documents d'utilisation des logiciels DeepFake	1 jour	Ven 27/09/19	Ven 27/09/19										
5			Rédaction du document sur l'état de l'art	1 jour	Ven 27/09/19	Ven 27/09/19										
6			Réunion avec le tuteur	1 jour	Mer 09/10/19	Mer 09/10/19										
7			Rédaction du cahier des charges	1 jour	Jeu 10/10/19	Jeu 10/10/19										
8			Validation par le tuteur pédagogique	1 jour	Ven 11/10/19	Ven 11/10/19										

Projet : avancement\_gantt\_pro

Date : Lun 16/03/20

Tâche

Fractionnement

Jalon

Récapitulative

Récapitulatif du projet

Tâche inactive

Jalon inactif

Récapitulatif inactif

Tâche manuelle

Durée uniquement

Report récapitulatif manuel

Récapitulatif manuel

Début uniquement

Fin uniquement

Tâches externes

Jalons externes

Échéance

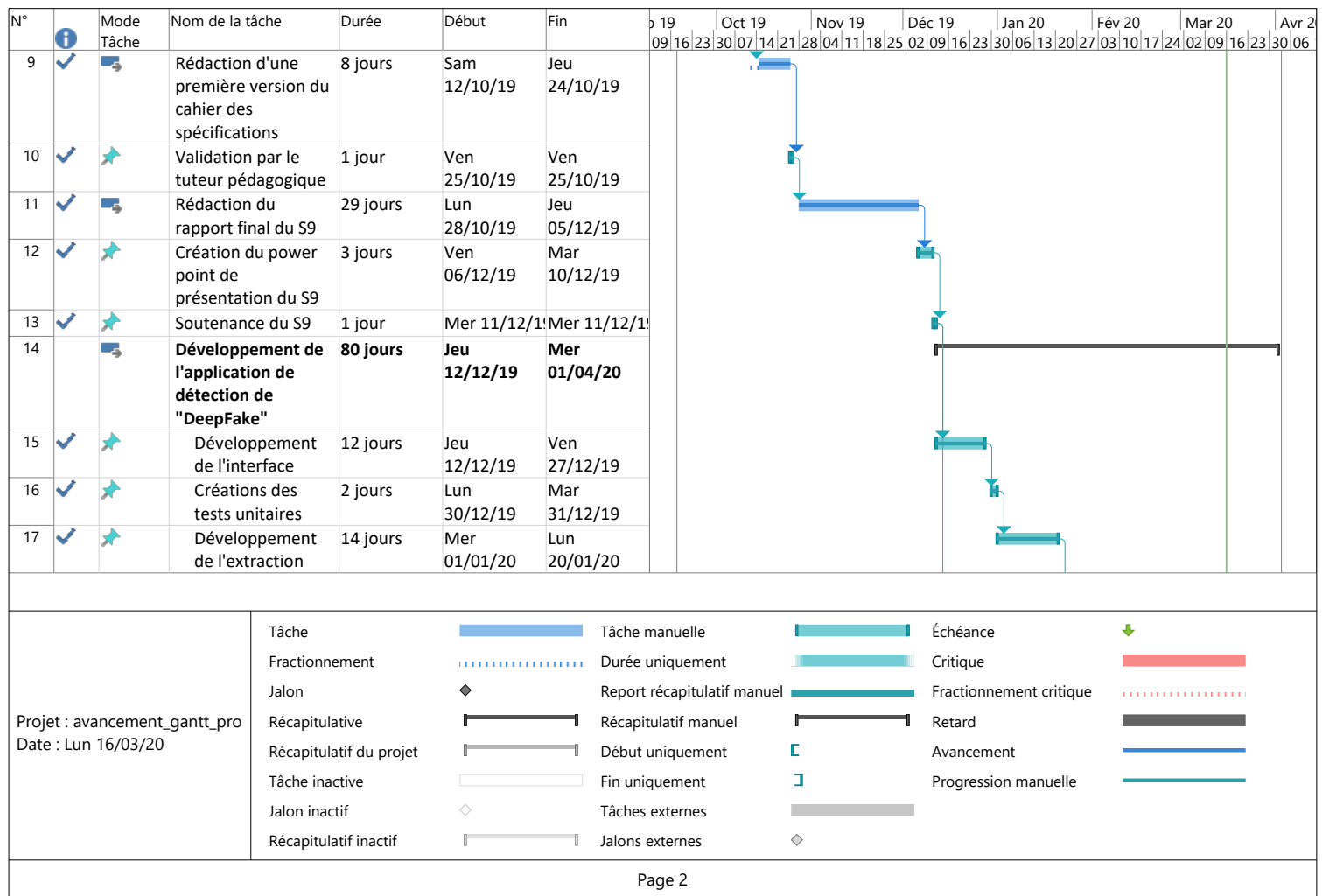
Critique

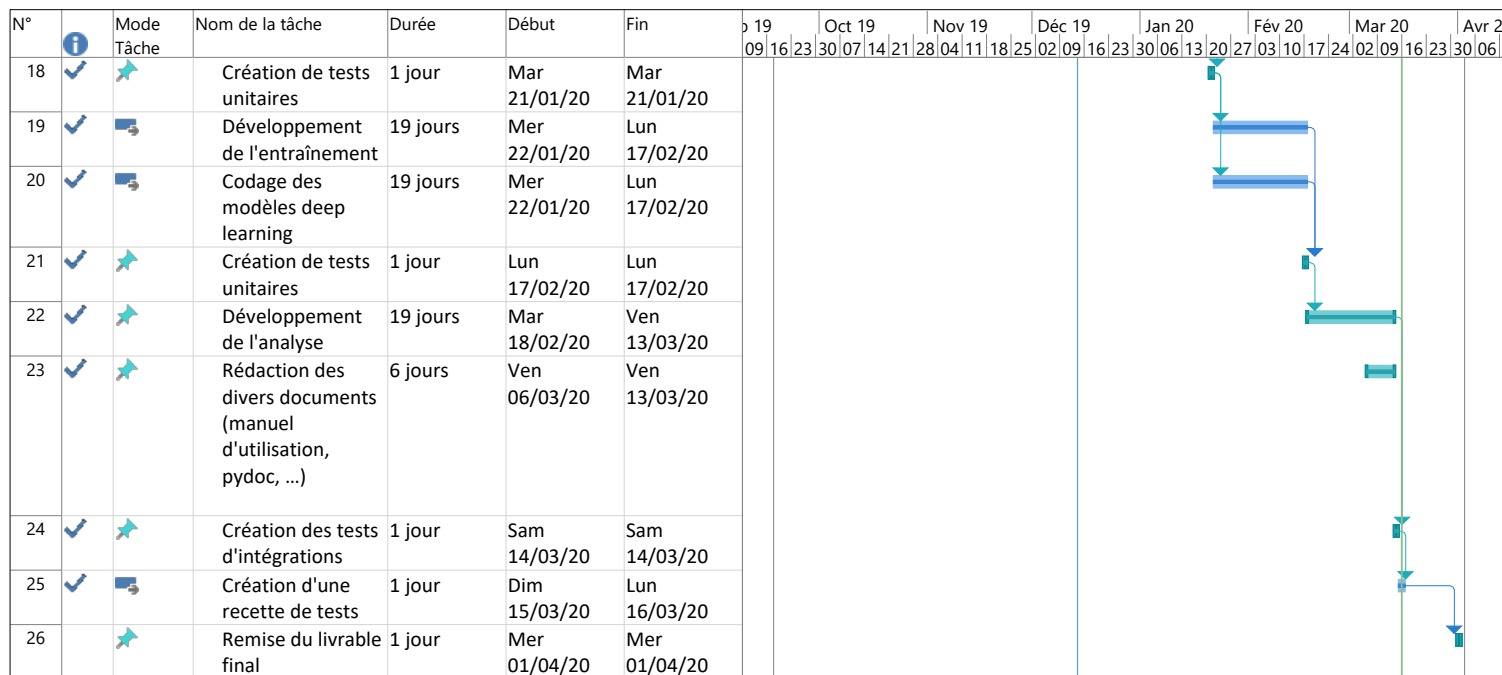
Fractionnement critique

Retard

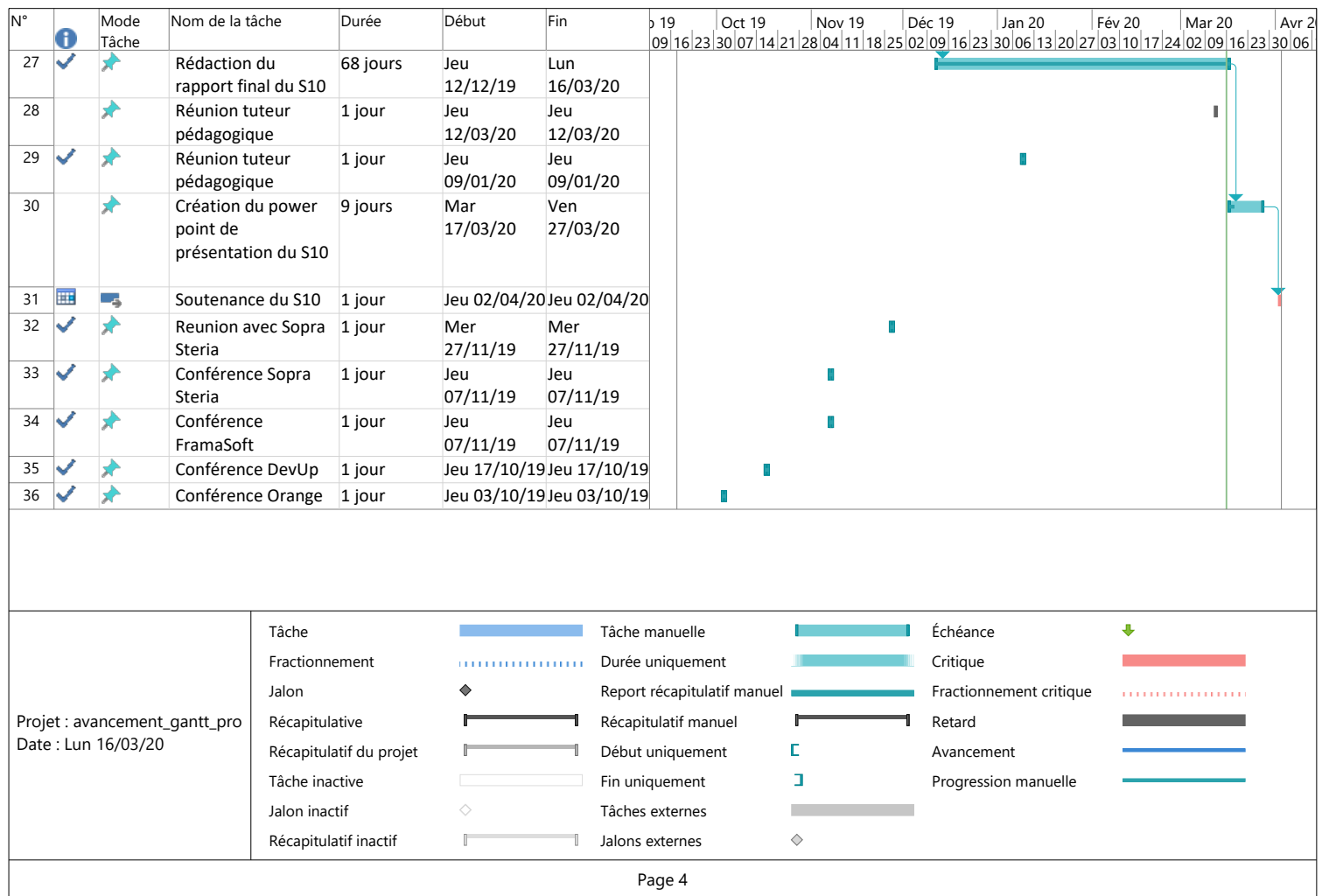
Avancement

Progression manuelle



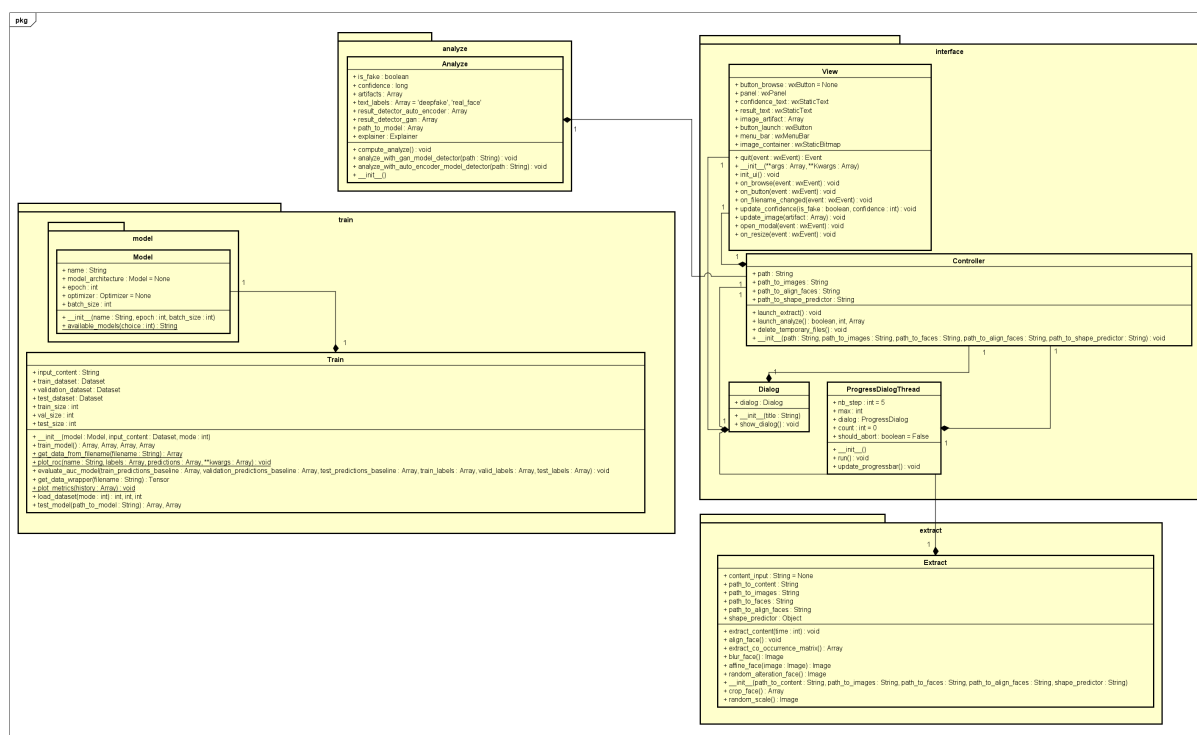


Projet : avancement_gantt_pro Date : Lun 16/03/20	Tâche		Tâche manuelle		Échéance	
	Fractionnement		Durée uniquement		Critique	
	Jalon		Report récapitulatif manuel		Fractionnement critique	
	Récapitulative		Récapitulatif manuel		Retard	
	Récapitulatif du projet		Début uniquement		Avancement	
	Tâche inactive		Fin uniquement		Progression manuelle	
	Jalon inactif		Tâches externes			
	Récapitulatif inactif		Jalons externes			



# Cahier du développeur

Ce document permet de décrire de manière détaillé chaque attribut, méthode et autres composants pour chaque classe du système.



**Figure 1 – Diagramme de classes du système**

- package : "analyze"
  - Classe "Analyze" :  
Description :  
Classe qui permet de réaliser l'analyse avec les modèles de deep learning implémentés et permettant de savoir si le contenu est falsifié ou non.
  - \* Attribut(s) :



1. `is_fake` : Boolean (catégorise la détection)
  2. `confidence` : Long (indicateur numérique de confiance dans l'hypothèse de détection)
  3. `artifacts` : Array (image qui contient les zones d'intérêt de l'analyse)
  4. `text_labels` : Array (contient le nom des classes)
  5. `result_detector_auto_encoder` : Array (contient le pourcentage de prédiction et la classe)
  6. `result_detector_gan` : Array (contient le pourcentage de prédiction et la classe)
  7. `path_to_model` : Array (contient le chemin des modèles pré-entraînés utilisés)
  8. `explainer` : Explain (contient l'objet qui permet la visualisation du neurone)
- \* Méthode(s) :
1. `compute_analyze()` : void  
Description :  
Cette méthode permet de lancer l'analyse GAN et auto-encodeur puis de faire la comparaison de celui qui a un indicateur numérique le plus important et qui selon un certain seuil, permet de dire si le contenu est un "DeepFake" ou non. Elle permet aussi de créer l'image avec les zones d'intérêt de l'analyse si le contenu se révèle être falsifié.
  2. `analyze_with_gan_model()` : long  
Description :  
Cette méthode permet d'analyser l'image avec le modèle de deep learning de détection GAN et de retourner la confiance de l'hypothèse d'une falsification dont les artefacts identifiés.
  3. `analyze_with_auto_encoder_model()` : long  
Description :  
Cette méthode permet d'analyser l'image avec le modèle de deep learning de détection auto-encodeur et de retourner la confiance de l'hypothèse d'une falsification dont les artefacts identifiés.
  4. `__init__()`  
Description :  
Constructeur de la classe Analyze
- package : "train"
    - Classe "Train" :  
Description :  
Permet d'entraîner les différents modèles de deep learning, charger un modèle pré-entraîné ou le sauvegarder sous forme de fichier.
- \* Attribut(s) :
1. `input_content` : String (chemin du contenu à utiliser)
  2. `train_dataset` : Dataset (jeu de données d'entraînement)
  3. `validation_dataset` : Dataset (jeu de données de validation)
  4. `test_dataset` : Dataset (jeu de données de test)
  5. `train_size` : int (taille du jeu de données d'entraînement)
  6. `val_size` : int (taille du jeu de données de validation)
  7. `test_size` : int (taille du jeu de données de test)
- \* Méthode(s) :
1. `__init__(model : Model, input_content : Dataset, mode : int)`  
Description :  
Constructeur de la classe Train
  2. `train_model()` : Array, Array, Array, Array  
Description :  
Cette méthode permet d'entraîner un modèle de deep learning spécifique.

3. get\_data\_from\_filename(filename : String) : Array  
Description :  
Cette méthode permet d'extraire la donnée et l'étiquette associée à un fichier compressé .npz.
4. plot\_roc(name : String, labels : Array, predictions : Array, \*\*kwargs : Array) : void  
Description :  
Cette méthode permet d'afficher les courbes AUC-ROC associées aux prédictions sur les différentes bases.
5. evaluate\_auc\_model(train\_predictions\_baseline : Array, validation\_predictions\_baseline : Array, test\_predictions\_baseline : Array, train\_labels : Array, valid\_labels : Array, test\_labels : Array) : void  
Description :  
Cette méthode permet de lancer les différentes évaluations avec les courbes ROC et la métrique AUC en fonction des jeux de données.
6. get\_data\_wrapper(filename : String) : Tensor  
Description :  
Cette méthode permet de récupérer les données venant des fichiers compressés .npz et de les transformer sous forme de tenseurs.
7. plot\_metrics(history : Array) : void  
Description :  
Cette méthode permet de visualiser les différentes courbes du modèle de deep learning (accuracy et loss).
8. load\_dataset(mode : int) : int, int, int  
Description :  
Cette méthode permet de charger en mémoire les différentes bases à destination du modèle de deep learning.
9. test\_model(path\_to\_model : String) : Array, Array  
Description :  
Cette méthode permet d'évaluer un modèle pré-entraîné enregistré sur une base de test.

— package "model"

1. Classe "Model" :

Description :

Contient les caractéristiques des modèles d'apprentissage profond qui seront utilisés pour la détection.

\* Attribut(s) :

- (a) name : String (nom du modèle)
- (b) model\_architecture : Model (configuration du modèle)
- (c) epoch : int (nombre de fois que le modèle doit recommencer l'entraînement sur les données)
- (d) optimizer : Optimizer (choix de la fonction de coût)
- (e) batch\_size (taille de l'échantillon de données en entrée du modèle)

\* Méthode(s) :

- (a) \_\_init\_\_(name : String, epoch : int, batch\_size : int)

Description :

Constructeur de Model

- (b) available\_models(choice : int) : String

Description :

Cette méthode permet de récupérer le nom du modèle correspondant à l'index du tableau

- package : "extract"

## — Classe "Extract" :

## Description :

Permet d'extraire du contenu les visages (image ou vidéo) pour l'utiliser dans l'analyse. Son autre rôle optionnel est de permettre aussi la création des exemples négatif pour l'auto-encodeur sans avoir recours à un logiciel de "DeepFake".

## \* Attribut(s) :

1. content\_input : String = None (image temporaire en cours de traitement)
2. path\_to\_content : String (chemin de l'entrée utilisateur)
3. path\_to\_images : String (chemin du fichier image)
4. path\_to\_faces : String (chemin du fichier visage)
5. path\_to\_align\_faces : String (chemin du fichier visage aligné)
6. shape\_predictor : Object (prédicteur de formes)

## \* Méthode(s) :

1. \_\_init\_\_(path\_to\_content : String, path\_to\_images : String, path\_to\_faces : String, path\_to\_align\_faces : String, shape\_predictor : String)  
Description :  
Constructeur de Extract
2. extract\_content(time : int) : void  
Description :  
Cette méthode permet de séparer en plusieurs images si l'entrée est une vidéo et d'extraire le contenu de chaque image, en découpant la zone d'intérêt du visage.
3. align\_face() : void  
Description :  
Cette méthode permet d'aligner le visage du contenu de façon à avoir une symétrie identique dans toutes les entrées.
4. extract\_co\_occurrence\_matrix() : Array  
Description :  
Cette méthode permet de calculer et d'obtenir la matrice de co-occurrence de l'image.
5. blur\_face() : Image  
Description :  
Cette méthode permet de flouter le visage contenu dans l'image.
6. affine\_face(image : Image) : Image  
Description :  
Cette méthode permet d'appliquer le visage flouté sur l'image originale afin d'obtenir une simulation d'un "DeepFake" par auto-encodeur.
7. random\_alteration\_face() : Image  
Description :  
Cette méthode permet d'appliquer sur l'image une altération aléatoire afin d'augmenter le nombre de possibilités dans le "DeepFake" possible du visage (contraste, couleur, échelle, finesse, ...).
8. crop\_face() : Array  
Description :  
Cette méthode permet d'extraire les ROI sur l'image, c'est à dire la zone du visage.
9. random\_scale() : Image  
Description :  
Cette méthode permet d'appliquer un redimensionnement à l'image.

## • package : "interface"

## — Classe "View" :

## Description :

Permet à l'utilisateur de pouvoir interagir avec le système par une interface graphique. La mise à jour des informations se fait par la classe "Controller".

\* Attribut(s) :

1. button\_browse : wxButton (rechercher un fichier)
2. button\_launch : wxButton (lancer l'analyse du contenu)
3. panel : wxPanel (panneau de l'interface)
4. image\_artifact : Array (Image sous forme matricielle)
5. confidence\_text : wxStaticText (représente la confiance)
6. result\_text : wxStaticText (représente la classe prédit)
7. image\_container : wxStaticBitmap (image de convolution d'un neurone)
8. menu\_bar : wxMenuBar (menu de la fenêtre)

\* Méthode(s) :

1. \_\_init\_\_(\*\*args : Array, \*\*Kwargs : Array)  
Description :  
Constructeur de View
2. quit(event : wxEvent) : Event  
Description :  
Cette méthode permet de fermer l'interface logiciel.
3. init\_ui() : void  
Description :  
Cette méthode permet d'initialiser l'interface.
4. on\_browse(event : wxEvent) : void  
Description :  
Cette méthode permet de lancer la recherche de fichier.
5. on\_button(event : wxEvent) : void  
Description :  
Cette méthode permet de lancer l'analyse du contenu.
6. on\_filename\_changed(event : wxEvent) : void  
Description :  
Cette méthode permet de vérifier l'existence du chemin.
7. update\_confidence(is\_fake : boolean, confidence : int) : void  
Description :  
Cette méthode permet de mettre à jour la confiance du modèle sur l'interface.
8. update\_image(artifact : Array) : void  
Description :  
Cette méthode permet de mettre à jour l'image contenant les artefacts de convolution du modèle sur l'interface.
9. open\_modal(event : wxEvent) : void  
Description :  
Cette méthode permet d'ouvrir une modale.
10. on\_resize(event : wxEvent) : void  
Description :  
Cette méthode permet de gérer l'affichage de la fenêtre lors de son redimensionnement.

— Classe "Controller" :

Description :

Permet à l'interface graphique de pouvoir intégrer les changements d'état du système au niveau du modèle.

\* Attribut(s) :

1. path : String (chemin du contenu utilisateur)
2. path\_to\_images : String (chemin du fichier image)

- 3. path\_to\_align\_faces : String (chemin du fichier visage aligné)
- 4. path\_to\_shape\_predictor : String (chemin du prédicteur de forme)
- \* Méthode(s) :
  - 1. \_\_init\_\_(path : String, path\_to\_images : String, path\_to\_faces : String, path\_to\_align\_faces : String, path\_to\_shape\_predictor : String)  
Description :  
Constructeur de Controller
  - 2. launch\_extract() : void  
Description :  
Cette méthode permet de lancer l'extraction du contenu.
  - 3. launch\_analyze() : void  
Description :  
Cette méthode permet de lancer l'analyse du contenu.
  - 4. delete\_temporary\_files() : void  
Description :  
Cette méthode permet de supprimer les fichiers temporaires.
- Classe "Dialog" :  
Description :  
Permet d'intégrer des modales afin d'aider l'utilisateur ou informer des changements d'état du système.
  - \* Attribut(s) :
    - 1. dialog : Dialog (modale)
  - \* Méthode(s) :
    - 1. \_\_init\_\_(title : String)  
Description :  
Constructeur de Dialog
    - 2. show\_dialog() : void  
Description :  
Cette méthode permet d'afficher une modale.
- Classe "ProgressDialogThread" :  
Description :  
Permet d'intégrer une modale de progression d'informer des changements d'état du système lors de l'extraction et analyse.
  - \* Attribut(s) :
    - 1. nb\_step : int = 5 (nombre d'étapes)
    - 2. max : int (maximum de la barre de progression)
    - 3. dialog : Dialog (modale)
    - 4. count : int = 0 (met à jour la progression)
    - 5. should\_abort : boolean = False (permet de stopper le thread)
    - 6. start : void() (démarré le thread)
  - \* Méthode(s) :
    - 1. \_\_init\_\_()  
Description : Constructeur de ProgressDialogThread
    - 2. run() : void  
Description :  
Cette méthode permet de créer les instructions du thread.
    - 3. update\_progressbar() : void  
Description :  
Cette méthode permet de mettre à jour la barre de progression.

A blue square containing a white capital letter 'F'.

## Cahier des charges

Le document ci-dessous constitue le regroupement des besoins exprimés par le client nécessaire à l'orientation du projet de recherche et développement.

09 OCTOBRE 2019

# CAHIER DES CHARGES

PROJET DE RECHERCHE ET DEVELOPPEMENT :  
LOGICIEL FAKENEWS : DETECTEUR D'IMAGES ET DE VIDEOS  
CONTREFAITES

ALEXANDRE BURNIER-FRAMBORET  
ELEVE INGENIEUR DE 5EME ANNEE A POLYTECH TOURS

## Table des matières

Contexte et définition du problème .....	2
Objectif du projet .....	2
Périmètre.....	2
Description fonctionnelle des besoins.....	2
Délai .....	4



## Contexte et définition du problème

Apparu initialement en 2008 comme un processus fictif, c'est en 2016 que les premières techniques de manipulation vidéo, notamment au niveau du visage apparaissent. Avec l'émergence de ces possibilités techniques, de nombreux logiciels open source sont apparues et permettent désormais à n'importe qui de pouvoir falsifier images et vidéos. Il s'agit donc d'un sujet d'actualité très récent, qui prend de plus en plus d'ampleur avec des techniques de plus en plus élaborées avec le temps. Ce processus est majoritairement utilisé pour la pornographie mais s'utilise aussi pour la falsification de contenu à visé politique, social voir de défense. De nombreux acteurs privés (Facebook Challenge, Google, ...) comme Etats se sont emparés du sujet par la recherche d'une détection efficace et automatique de ces contenus afin de prévenir d'éventuelle effets négatifs à la suite de leur diffusion sur internet.

## Objectif du projet

L'objectif du projet s'articule autour de la problématique de la détection d'images ou vidéos falsifiées par les techniques de l'apprentissage profond communément appelée « DeepFake ».

## Périmètre

Le périmètre du projet se concentrera uniquement sur les images et vidéos contenant des individus, ayant leur visage modifié par le deep learning ou crée à partir d'une génération artificielle par un réseau de neurones de type GAN (generative adversarial networks) qui permet la création de contenu à fort réalisme.

## Description fonctionnelle des besoins

Fonction principale	Savoir si une image d'un individu est falsifiée
Description	Permettre à l'utilisateur de savoir selon un indicateur si l'image en entrée est une image falsifiée ou non
Sous fonctions	Prendre une image en entrée Analyser l'image Afficher le résultat de la décision
Contraintes / règles de gestion	Image générée par un réseau de neurones de type GAN ou auto-encodeur (remplacement du visage)
Priorité	Haute

Sous fonction	Prendre une image en entrée
Description	Permettre à l'utilisateur de pouvoir utiliser une image en entrée
Sous fonctions	/
Contraintes / règles de gestion	Aucune
Priorité	Haute

Sous fonction	Analyser l'image
Description	Analyser l'image à l'aide de techniques issues de l'apprentissage profond tel que CNN, LSTM...
Sous fonctions	/

<b>Contraintes / règles de gestion</b>	Aucune
<b>Priorité</b>	Haute

<b>Sous fonction</b>	<b>Afficher le résultat de la décision</b>
<b>Description</b>	Permettre à l'utilisateur de connaître la décision du réseau de neurone sous la forme d'un indicateur correspondant à celui-ci
<b>Sous fonctions</b>	/
<b>Contraintes / règles de gestion</b>	Aucune
<b>Priorité</b>	Haute

<b>Fonction principale</b>	<b>Savoir si une vidéo d'un individu est falsifiée</b>
<b>Description</b>	Permettre à l'utilisateur de savoir selon un indicateur si la vidéo est falsifiée
<b>Sous fonctions</b>	Prendre une vidéo en entrée Analyser la vidéo Afficher le résultat de la décision
<b>Contraintes / règles de gestion</b>	Vidéo générée par un réseau de neurones de type GAN ou auto-encodeur (remplacement du visage)
<b>Priorité</b>	Haute

<b>Sous fonction</b>	<b>Prendre une vidéo en entrée</b>
<b>Description</b>	Permettre à l'utilisateur de pouvoir utiliser une vidéo en entrée
<b>Sous fonctions</b>	/
<b>Contraintes / règles de gestion</b>	Aucune
<b>Priorité</b>	Haute

<b>Sous fonction</b>	<b>Analyser la vidéo</b>
<b>Description</b>	Analyser la vidéo à l'aide de techniques issues de l'apprentissage profond tel que CNN, LSTM...
<b>Sous fonctions</b>	/
<b>Contraintes / règles de gestion</b>	Aucune
<b>Priorité</b>	Haute

<b>Sous fonction</b>	<b>Afficher le résultat de la décision</b>
<b>Description</b>	Permettre à l'utilisateur de connaître la décision du réseau de neurone sous la forme d'un indicateur correspondant à celui-ci
<b>Sous fonctions</b>	/
<b>Contraintes / règles de gestion</b>	Aucune
<b>Priorité</b>	Haute

<b>Fonction principale</b>	<b>Montrer les zones de détections des artefacts dans l'image ou la vidéo</b>
<b>Description</b>	Permettre à l'utilisateur de constater visuellement les zones de la détection qui permirent de décider du jugement de la vidéo
<b>Sous fonctions</b>	Afficher les artefacts de l'image ou de la vidéo
<b>Contraintes / règles de gestion</b>	Aucune
<b>Priorité</b>	Moyenne

<b>Sous fonction</b>	<b>Afficher les artefacts de l'image ou de la vidéo</b>
<b>Description</b>	Permettre à l'utilisateur de pouvoir visualiser les points de détection de l'image ou de la vidéo
<b>Sous fonctions</b>	/
<b>Contraintes / règles de gestion</b>	Aucune
<b>Priorité</b>	Haute

## Délai

Le délai principal du projet correspond au rendu de la dernière partie du rapport final, soit une semaine avant sa présentation, début avril. La première partie du rapport final sera rendu le 5 décembre et une première version du cahier des spécifications devra être rendu avant le 31 octobre 2019.



## Manuels d'utilisation

Les documents ci-dessous constituent les manuels d'utilisation des logiciels de "DeepFake" utilisés dans le cadre de ce projet de recherche et développement ainsi que ceux à destination de l'utilisateur et développeur.

# Tutoriel de prise en main du logiciel Face2Face

Alexandre Burnier-Framboret

25 septembre 2019

# Introduction

Face2Face est un logiciel dit de création de "DeepFake" qui permet à l'utilisateur de pouvoir, à l'aide d'une webcam de pouvoir créer une réplique vidéo d'une autre personne à travers ses expressions faciales. Il est ainsi aisé de pouvoir créer une "FakeNews" en usurpant l'identité d'une autre personne.

## Tutoriel de prise en main

Après extraction du projet `git@github.com:datitran/face2face-demo.git`  
Pour utiliser le logiciel :

1. Une vidéo en entrée à convertir en "DeepFake"
2. Un modèle "frozen" entraîné qui correspond à la personne de la vidéo
3. Un shape predictor qui correspond à la modélisation des caractéristiques du visage ...

## Création de la donnée d'entraînement

```
python generate_train_data.py -file angela_merkel_speech.mp4  
-num 400 -landmark-model shape_predictor_68_face_landmarks.dat
```

Un fichier original et landmarks seront créés à la suite de la commande.

## Entraînement du modèle avec les données d'entraînement

On utilisera `pix2pix-tensorflow` afin de créer le modèle :  
`git clone https://github.com/affinelayer/pix2pix-tensorflow.git`

Ensuite, il faut déplacer les fichiers originaux et landmarks précédemment créés dans le répertoire `pix2pix-tensorflow.git` et faire les commandes suivantes :

### **Redéfinir la taille des images originales**

```
python tools/process.py --input_dir photos/original --operation resize --output_dir
photos/original.resized
```

### **Redéfinir la taille des images avec landmarks**

```
python tools/process.py --input_dir photos/landmarks --operation resize --output_dir
photos/landmarks.resized
```

### **Combiner les deux images originales et landmarks**

```
python tools/process.py --input_dir photos/landmarks.resized --b_dir photos/original.resized
--operation combine --output_dir photos/combined
```

### **Séparer les données en entraînement et validation**

```
python tools/split.py --dir photos/combined
```

### **Entraîner le modèle (cela peut prendre plusieurs heures)**

```
python pix2pix.py --mode train --output_dir face2face-model --max_epochs 200
--input_dir photos/combined/train --which_direction AtoB
```

### **Réduire le modèle pour être plus léger lors de l'utilisation**

```
python reduce_model.py --model-input face2face-model --model-output face2face-
reduced-model
```

On freeze ensuite le modèle pour être plus léger lors de l'utilisation

```
python freeze_model.py --model-folder face2face-reduced-model
```

### **Lancement du programme (la caméra doit être disponible)**

```
python run_webcam.py --source 0 --show 0 --landmark-model shape_predictor_exemple.dat
--tf-model frozen_model.pb
```

## Appendix A

### Annexe

Face2Face : <https://github.com/datitran/face2face-demo>

pix2pix-tensorflow : <https://github.com/affinelayer/pix2pix-tensorflow>



# Tutoriel de prise en main du logiciel FaceSwap

Alexandre Burnier-Framboret

25 septembre 2019

# Introduction

FaceSwap est un logiciel dit de création de "DeepFake" qui permet à l'utilisateur de pouvoir, à l'aide d'une vidéo ou d'une image de pouvoir créer une réplique vidéo ou une image d'une autre personne à travers ses expressions faciales. De nombreux formats et outils sont disponibles pour la méthode de création du visage. Il est donc devenu aisé de pouvoir créer une "FakeNews" en usurpant l'identité d'une autre personne.

## Tutoriel de prise en main

Après extraction du projet <https://github.com/deepfakes/faceswap.git>

Pré-requis nécessaire pour utiliser le logiciel :

1. Une vidéo ou banque de donnée d'images d'une seule personne en entrée à utiliser comme cible
2. Une autre vidéo ou une banque de données d'images d'une seule personne en entrée à utiliser pour transposer son visage sur celui de la cible
3. (Facultatif) Un modèle pré-entraîné qui correspond à la fusion entre les deux visages

Il existe deux modes pour utiliser le logiciel :

1. En ligne de commande
2. En GUI (graphical user interface)

**Veillez à faire `python setup.py` avant l'utilisation afin de garantir les fonctionnalités optimales du logiciel.**

## Mode ligne de commande

### Extraction des visages venant des images

#### Depuis un fichier de photos

```
python faceswap.py extract -i /faceswap/src/trump -o /faceswap/faces/trump  
python faceswap.py extract -i /faceswap/src/cage -o /faceswap/faces/cage
```

## Depuis une vidéo

```
python faceswap.py extract -i /faceswap/src/trump.mp4 -o /faceswap/faces/trump
python faceswap.py extract -i /faceswap/src/cage.mp4 -o /faceswap/faces/cage
```

### Options de l'extraction

- Détecteur : `-D cv2-dnn,mtcnn,s3fd, -detector cv2-dnn,mtcnn,s3fd`
  - Cv2-DNN
  - MTCNN
  - S3Fd
- Alignement : `-A cv2-dnn,fan, -aligner cv2-dnn,fan`
  - Cv2-DNN
  - Fan
- Normalisation : `-nm none,clahe,hist,mean, -normalization none,clahe,hist,mean`
  - None
  - Clahe
  - Hist
  - Mean
- Rotation de l'image : `-r ROTATE_IMAGES, -rotate-images ROTATE_IMAGES`
- Calcul du visage
  - Filtrer par taille du visage en pixel : `-min MIN_SIZE, -min-size MIN_SIZE`
  - Filtrer des visages non désirés : `-n NFILTER [NFILTER ...], -nfilter NFILTER [NFILTER ...]`
  - Filtrer plusieurs personne sur l'image : `-f FILTER [FILTER ...], -filter FILTER [FILTER ...]`
  - Seuil de reconnaissance pour l'extraction : `-l REF_THRESHOLD, -ref_threshold REF_THRESHOLD`
  - Seuil de floutage pour laquelle l'image ne doit pas être pris en compte : `-bt BLUR_THRESH, -blur-threshold BLUR_THRESH`
- Sortie
  - Extraction de l'image toute les N frames : `-een EXTRACT_EVERY_N, -extract-every-n EXTRACT_EVERY_N`
  - resolution de l'image de sortie (256\*256 par défaut) : `-sz SIZE, -size SIZE`
  - Sauvegarder l'alignement toute les N frames : `-si SAVE_INTERVAL, -save-interval SAVE_INTERVAL`
  - aligner la hauteur des yeux entre toute les images : `-ae, -align-eyes`

## Entraînement du modèle avec les données d'entraînement

```
python faceswap.py train -A /faceswap/faces/trump -B /faceswap/faces/cage  
-m /faceswap/trump_cage_model/
```

### Options de l'entraînement

- Entraînement
  - Dfaker
  - Dfl-H128
  - Dfl-Sae
  - lae
  - LightWeight
  - Original
  - Realface
  - Unbalanced
  - Villain
- Entraînement
  - Batch Size
  - Iterations
  - No Logs
  - Warp To Landmarks
  - No Flip
  - No Augment Color
- Sauvegarde
  - Save Interval
  - Snapshot Interval
- Timelapse
- Preview
  - Preview Scale
  - Preview
  - Write Image

```
[-C CONFIGFILE] [-L INFO,VERBOSE,DEBUG,TRACE] [-LF LOGFILE] -A  
INPUT_A [-ala ALIGNMENTS_PATH_A] -B INPUT_B [-alb ALIGNMENTS_PATH_B]  
-m MODEL_DIR [-t dfaker,df-h128,df-sae,iae,lightweight,original,realface,unbalanced,villain]  
[-bs BATCH_SIZE] [-it ITERATIONS] [-s SAVE_INTERVAL] [-ss SNAPSHOT_INTERVAL]  
[-tia TIMELAPSE_INPUT_A] [-tib TIMELAPSE_INPUT_B] [-to TIMELAPSE_OUTPUT]  
[-ps PREVIEW_SCALE] [-p] [-w] [-nl] [-wl] [-nf] [-nac]
```

## Lancer la conversion de l'image ou de la vidéo

```
python faceswap.py convert -i /faceswap/src/trump/ -o /faceswap/converted/  
-m /faceswap/trump_cage_model/
```

### Options de la conversion

- Ajustement de couleur
  - None
  - Avg-Color
  - Color-Transfer
  - Manual-Balance
  - Match-Hist
  - Seamless-Clone
- Type de masque
  - Components
  - Dfl Full
  - Extended
  - Facehull
  - None
  - Predited
- Mise a l'échelle
  - None
  - Sharpen
- Ecriture
  - Ffmpeg : Utiliser pour convertir une vidéo en entrée en vidéo en sortie
  - Gif : Utiliser pour créer un Gif en sortie
  - Opencv : Utiliser pour convertir une vidéo en entrée en images en sortie
  - Pillow : Plus lent que OpenCV mais supporte plus de format en entrée
- Calcul de l'image
  - Echelle de sortie
  - Frame Ranges

- Calcul du visage
  - Filtrer par taille du visage en pixel
  - Filtrer des visages non désirés
  - Filtrer plusieurs personne sur l'image
  - Seuil de reconnaissance pour l'extraction
  - Seuil de floutage pour laquelle l'image ne doit pas être pris en compte
- Paramètres
  - Jobs
  - Trainer : Donner le nom du modèle entraîné si plusieurs dans le même répertoire
  - Swap Model
  - SingleProcess

```
[-L INFO,VERBOSE,DEBUG,TRACE] [-LF LOGFILE] -i INPUT_DIR -o OUTPUT_DIR [-al ALIGNMENTS_PATH] [-ref REFERENCE_VIDEO] -m MODEL_DIR [-c none,avg-color,color-transfer>manual-balance,match-hist,seamless-clone] [-M components,dfl_full,extended,facehull,none,predicted] [-sc none,sharpen] [-w ffmpeg,gif,opencv,pillow] [-osc OUTPUT_SCALE] [-fr FRAME_RANGES [FRAMEi_RANGES ...]] [-a INPUT_ALIGNED_DIR] [-n NFILTER [NFILTER ...]] [-f FILTER [FILTER ...]] [-l REF_THRESHOLD] [-j JOBS]
```

## Mode GUI

python swapface.py gui

## Outils

### Alignments tool

Permet d'aligner un ensemble de visage sur les images afin d'obtenir une meilleure qualité d'entraînement.

### Effmpeg tool

Permet d'extraire les vidéos sous formes de collections d'images avec différents paramètres possible comme le format d'extraction, l'échelle, etc... Très utile pour créer rapidement des données d'entraînement.

### Preview

Permet de paramétrer la preview du modèle généré.

**Restore**

Permet de restaurer un modèle venant d'un backup (.bk).

**Sort**

Permet de trier les visages selon de multiples paramètres.

## Appendix A

### Annexe

FaceSwap : <https://github.com/deepfakes/faceswap.git>





# **MANUEL D'UTILISATION DU LOGICIEL « DEEPFAKE- DETECTOR »**

*Version du document : 1.0*

*Date de révision : 21/02/2020*

*Auteur : Alexandre Burnier-Framboret*

## Table des matières

A propos de l'application « DeepFake-Detector » .....	3
Configuration requise .....	3
Utiliser l'application « DeepFake-Detector » .....	3
Aide.....	7
Éléments de l'interface.....	8

## A propos de l'application « DeepFake-Detector »

Ce logiciel a pour but la détection de « DeepFake » au sein d'images ou vidéos contenant un individu.

Il permet entre autre de connaître à l'aide de l'IA si celui-ci est issu d'une falsification ou bien est authentique. Un pourcentage est donné à l'issue de l'analyse afin de situer la confiance dans la conclusion de l'analyse. Une image issue d'une des couches du réseau de neurones est affichée permettant d'avoir un caractère plus « explicatif » dans la décision.

## Configuration requise

### **Configuration minimum :**

- Processeur Intel Core i3 (équivalent AMD)
- Intel UHD Graphics
- 8 Go de RAM

### **Configuration recommandée :**

- Processeur Intel Core i7 (équivalent AMD)
- Carte graphique NVIDIA RTX 2060
- 32 Go

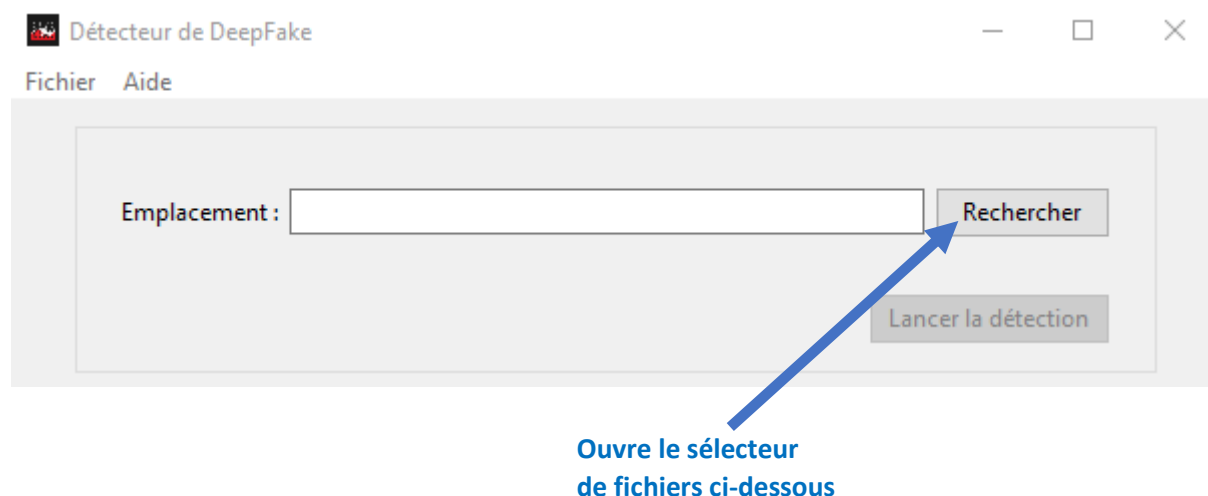
## Utiliser l'application « DeepFake-Detector »

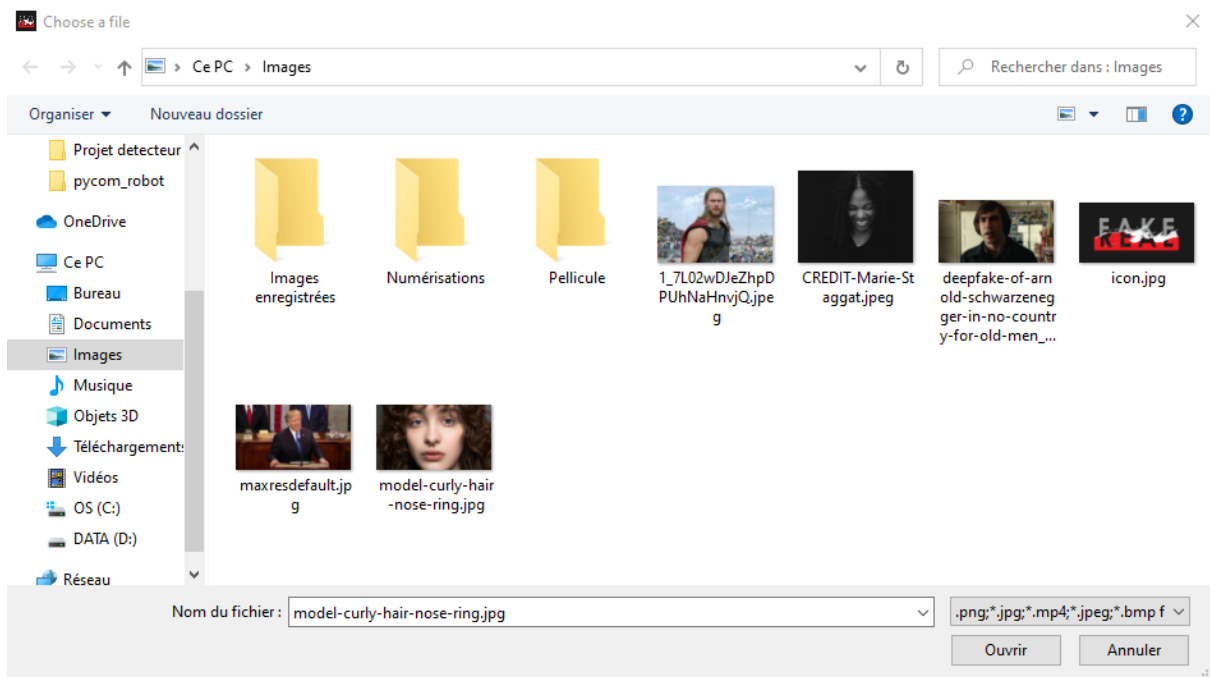
*Cette partie du guide est à but « tutoriel » avec une mise en œuvre pas-à-pas.*

### 1. Rechercher le contenu à analyser

La barre de recherche permet de rentrer / vérifier son chemin de fichier. Cette recherche peut aussi être faite à partir du bouton « Rechercher ».

**Attention :** Le contenu ne doit contenir uniquement que 1 seul individu.



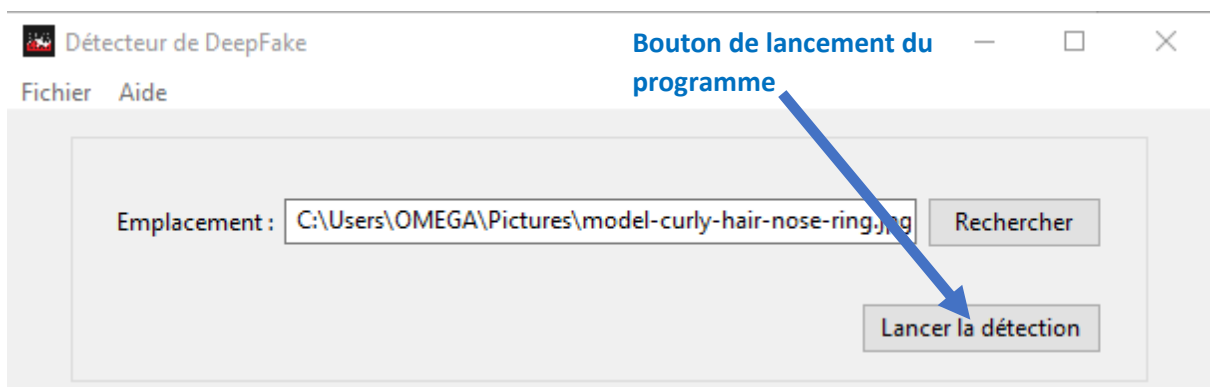


Les fichiers autorisés sont de format :

- .png
- .jpeg
- .jpg
- .mp4
- .bmp
- .avi

## 2. Lancer l'analyse du contenu

Lorsque le contenu a été saisi et que **le chemin est valide** ; le bouton « Lancer la détection » devient actif afin de pouvoir lancer l'analyse du programme.



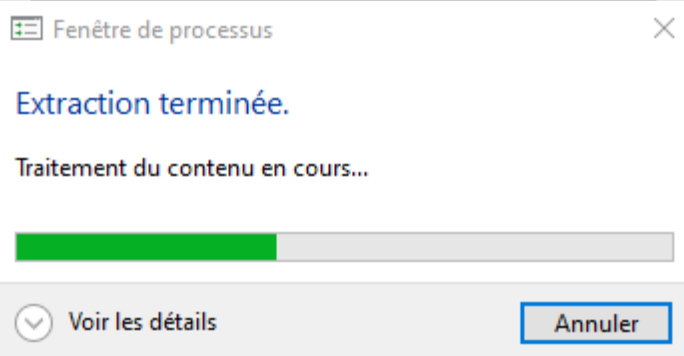
Après lancement de l'analyse, un certain temps (quelques minutes maximum) est nécessaire afin de prédire les informations nécessaires sur le contenu en entrée.

Emplacement : C:\Users\OMEGA\Pictures\model-curly-hair-nose-ring.jpg

Rechercher

Lancer la détection

### Artefacts identifiés dans le contenu

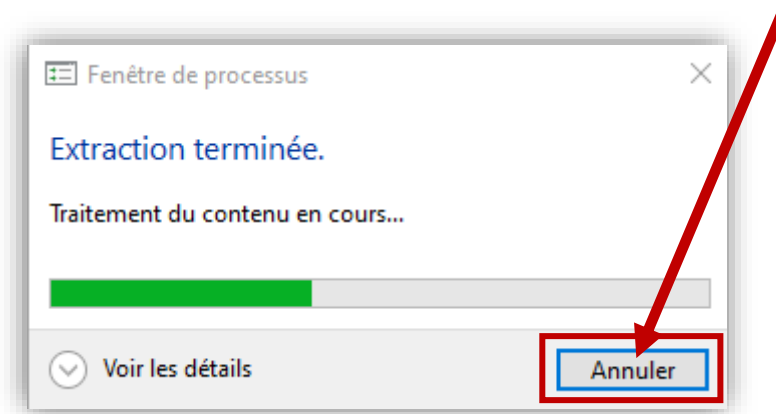


### Résultat de la détection

Status du contenu :

Confiance dans la détection (en %) :

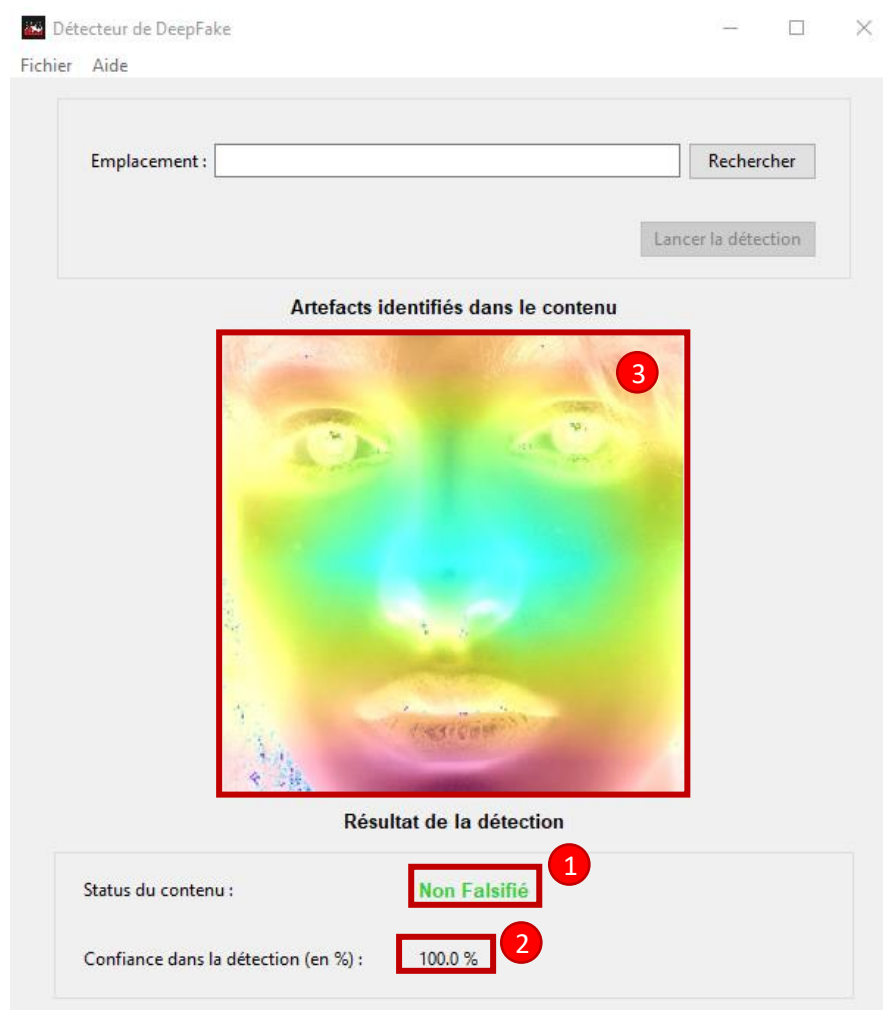
A tout moment l'analyse peut-être interrompu grâce au bouton « Annuler ».



### 3. Visualiser les différents éléments à la suite de l'analyse

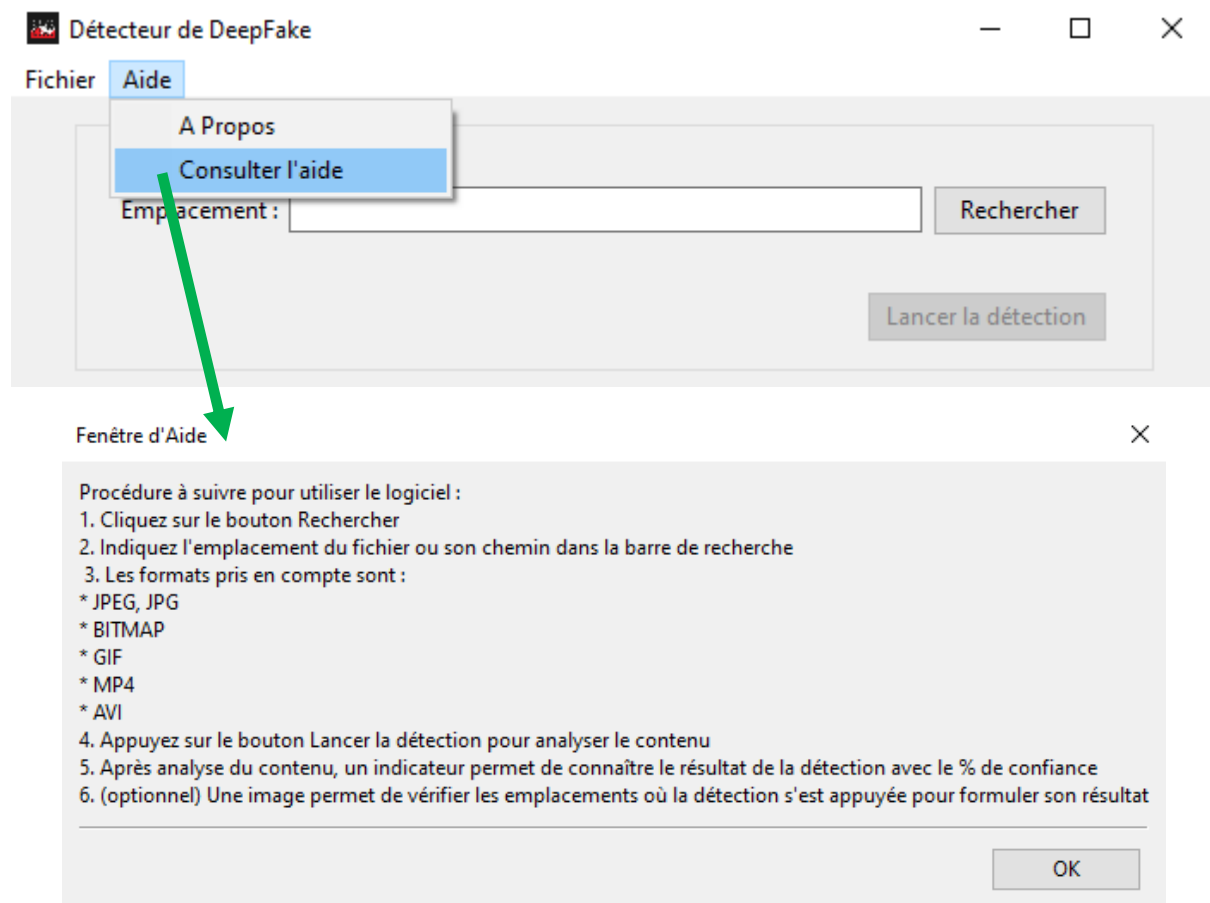
Après analyse du contenu, de nouvelles informations sont présentes tel que :

- La prédiction du réseau de neurones sur le contenu <sup>1</sup>
- La confiance dans la prédiction du réseau de neurones <sup>2</sup>
- Les artefacts de l'image qui ont permis au réseau de conclure <sup>3</sup>

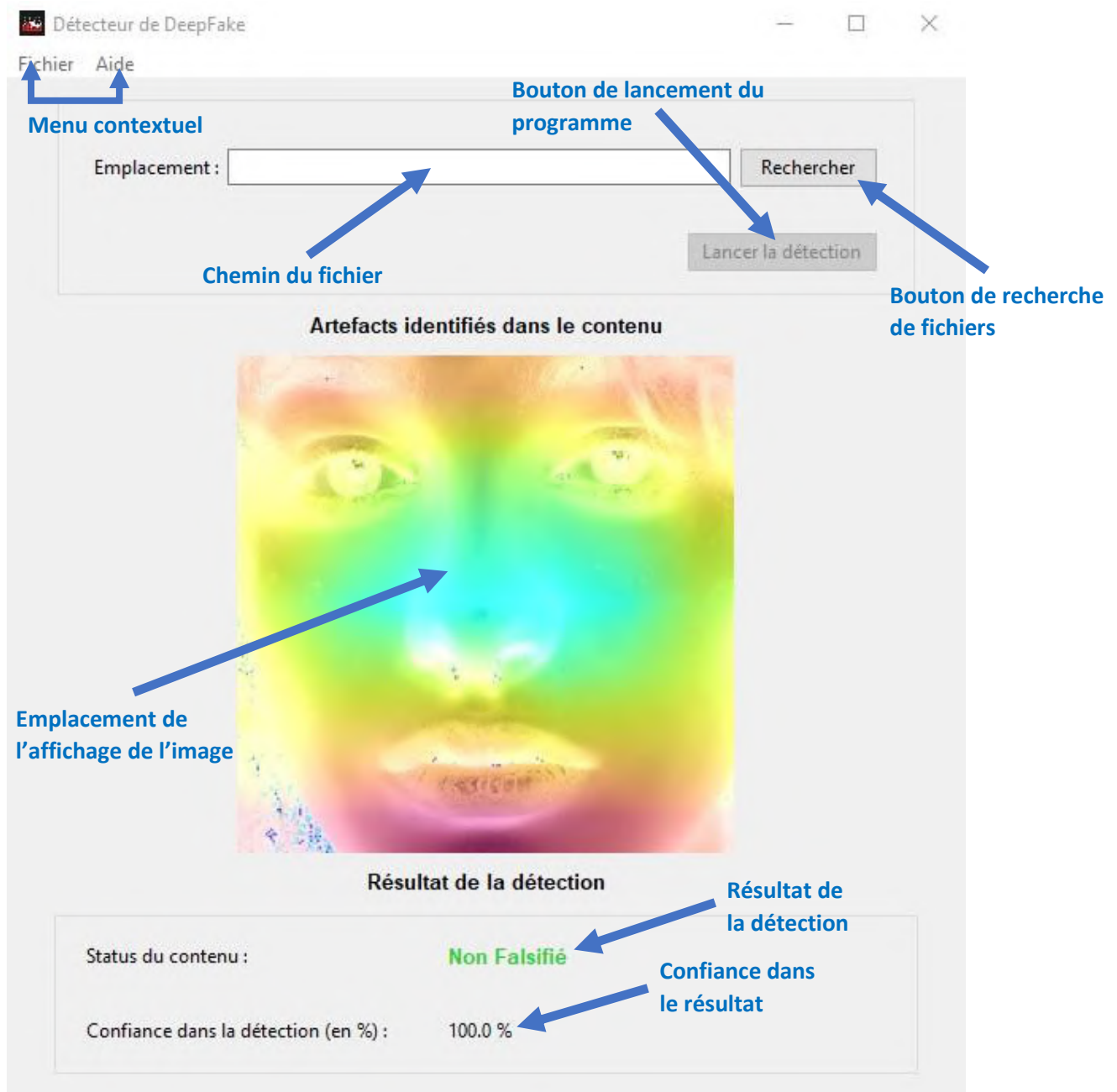


## Aide

Une aide est disponible dans le menu contextuel afin de rappeler la mise en œuvre du logiciel.



## Éléments de l'interface







# **MANUEL DU DEVELOPPEUR LOGICIEL « DEEPFAKE- DETECTOR »**

*Version du document : 1.0*

*Date de révision : 21/02/2020*

*Auteur : Alexandre Burnier-Framboret*

## Table des matières

A propos du manuel du développeur.....	3
Ensemble des diagrammes.....	3
Hiérarchie du code logiciel.....	8
Fichiers temporaires.....	9
Tester le code logiciel .....	9
Script de pré-traitement de la donnée .....	9
Script de gestion des entraînements/tests des modèles.....	9
Ajouter un modèle d'apprentissage profond .....	10

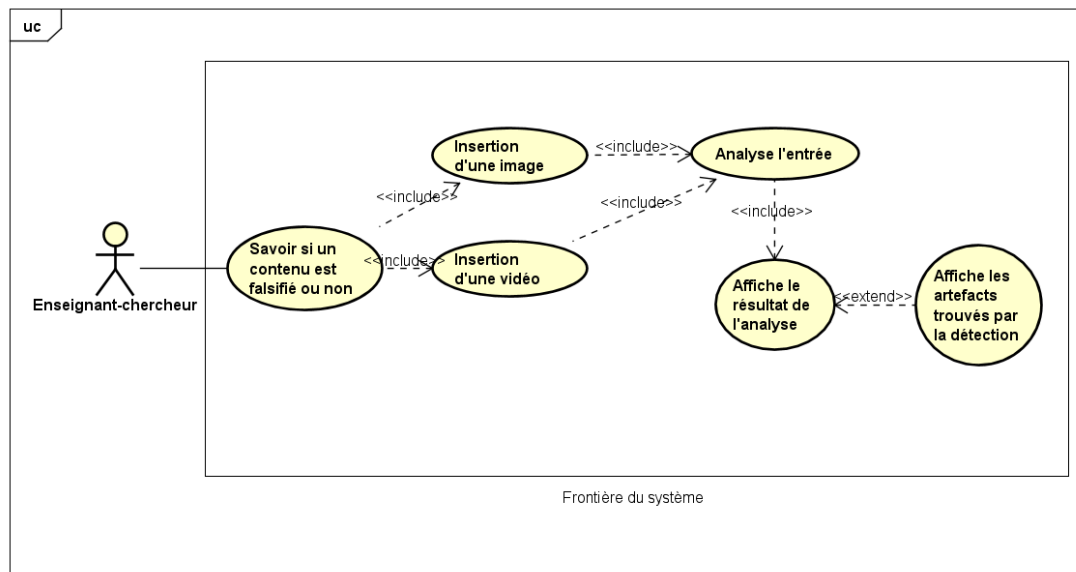
## A propos du manuel du développeur

*Ce manuel a pour but à défaut d'aider à la compréhension du fonctionnement du logiciel dans un cas de maintenance ou d'une éventuelle mise à jour.*

Le code open-source est disponible à l'adresse :

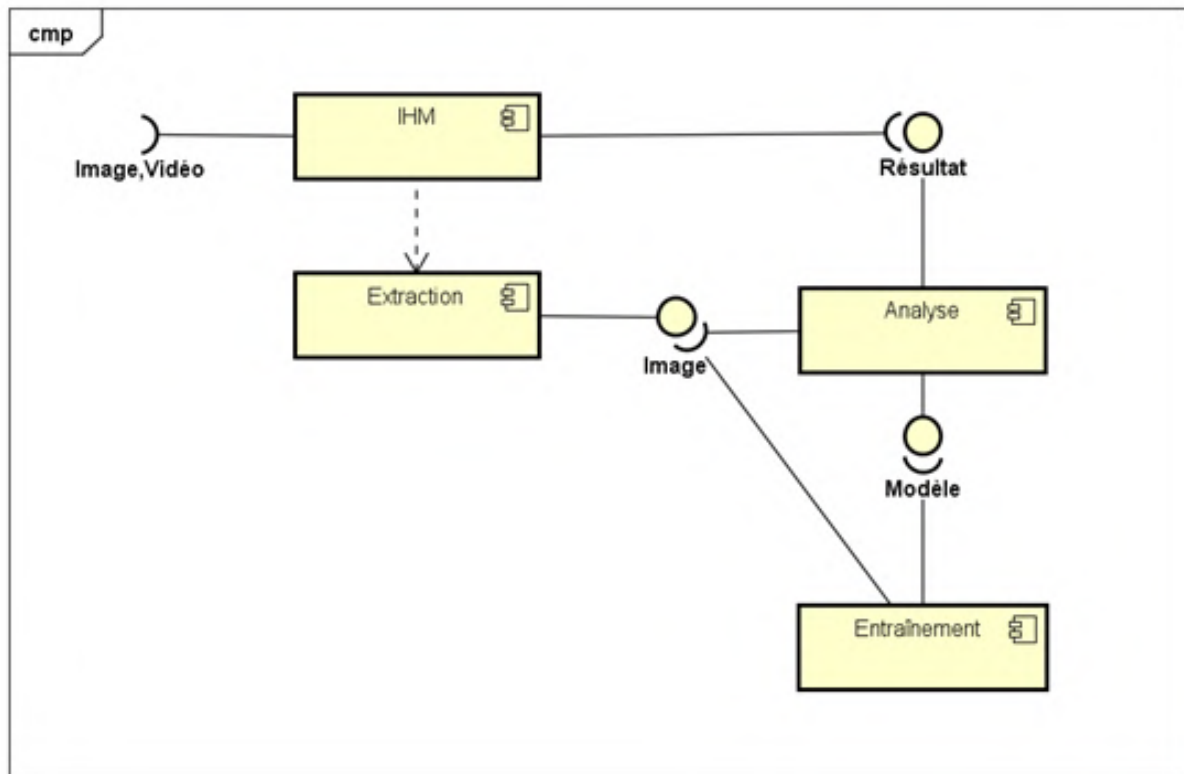
<https://gitlab.com/Overlorde/deepfake-detector.git>

## Ensemble des diagrammes



*Figure 1 - Diagrammes des cas d'utilisations*

Le logiciel a pour but de prendre en entrée une image ou vidéo, d'analyser son contenu et fournir en sortie une décision sur celui-ci avec éventuellement les artefacts de l'image qui ont permis au réseau de neurones de conclure.



powered by Astah

*Figure 2 - Diagramme de composants*

Ce diagramme permet de synthétiser l'architecture logiciel. L'IHM utilise le composant « Extraction » qui fournit une Image en entrée au composant « Analyse » qui utilise le Modèle du composant « Entraînement » afin de fournir un « Résultat » en retour à l'IHM.

Ainsi, le composant « Entraînement » ne sert que à fournir un modèle déjà pré-entraîné. Pour fournir d'autres type de Modèle au composant « Analyse », il faut dans ce cas apporter des modifications au composant Modèle en ajoutant de nouveaux modèles (architecture) dans la partie correspondante du code. Le composant Entraînement n'a pas besoin d'ajout de modifications, étant suffisamment flexible dans son adaptation au modèle.

Le choix de ne pas avoir la possibilité d'utiliser le composant Entraînement par l'interface est dû à l'utilisateur potentiel qui n'est pas un expert IA et pour lequel l'apport de cette fonctionnalité serait inutile.

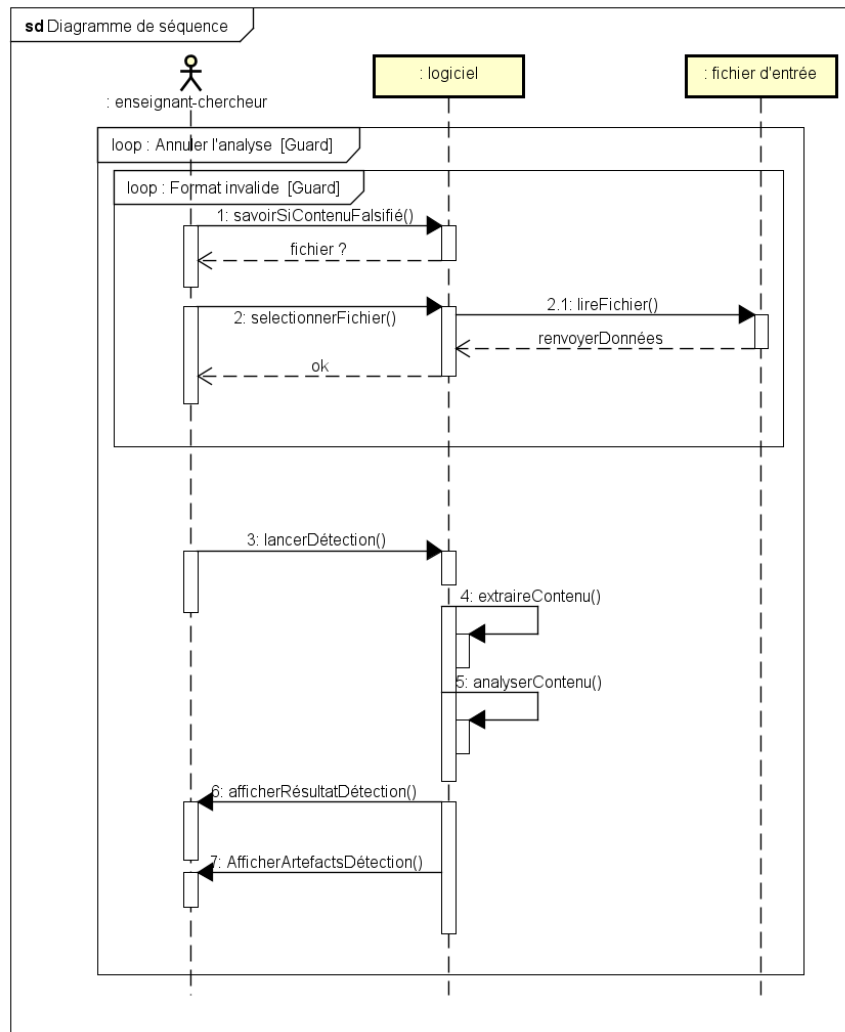


Figure 3 - Diagramme de séquence

Le diagramme ci-dessus permet de résumer le cycle de vie du logiciel. Ainsi, l'utilisateur rentre dans le système son entrée, le logiciel vérifie le format, si celle-ci est incorrecte, il doit redonner une entrée sinon le logiciel analyse le contenu et affiche ensuite les différents résultats et l'image à l'utilisateur.

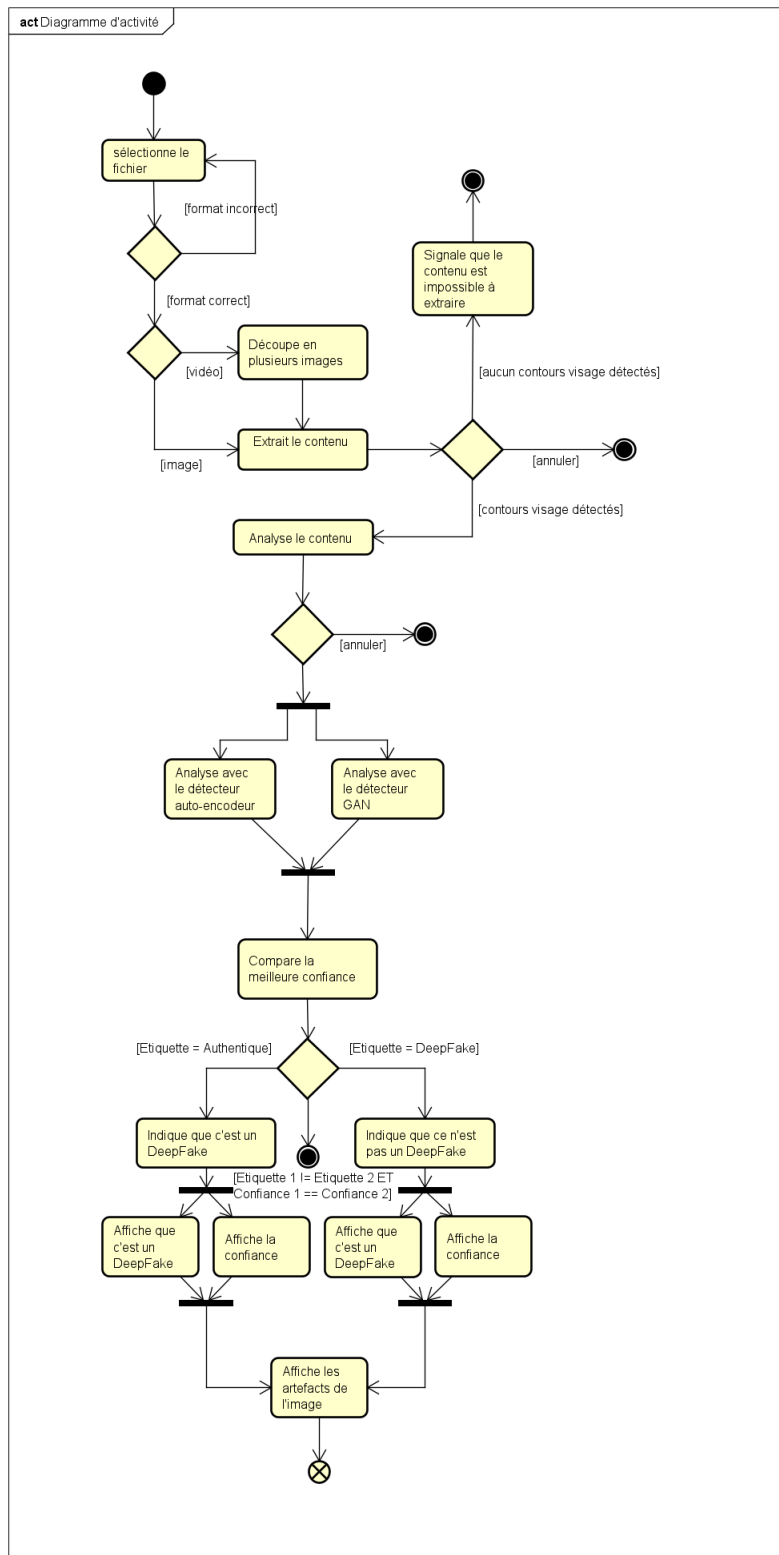


Figure 4 - Diagramme d'activité

Le diagramme d'activité ci-dessus permet de comprendre en détail les différentes étapes du logiciel, de l'entrée utilisateur à la sortie logiciel.



- Cette classe contient la modale de progression permettant à l'utilisateur de connaître la progression dans l'exécution de l'analyse du logiciel.
- Classe View :
  - Cette classe contient les éléments de l'IHM et interagit avec le Controller pour effectuer les actions tel que l'extraction et l'analyse.
- Classe Controller :
  - Cette classe contient les différentes actions effectués par le logiciel lors de l'analyse c'est-à-dire extraire les informations utile pour pouvoir ensuite les analyser avec les modèles implémentées.

Plus de détails sur les classes sont disponibles dans la PyDoc fourni.

## Hiérarchie du code logiciel

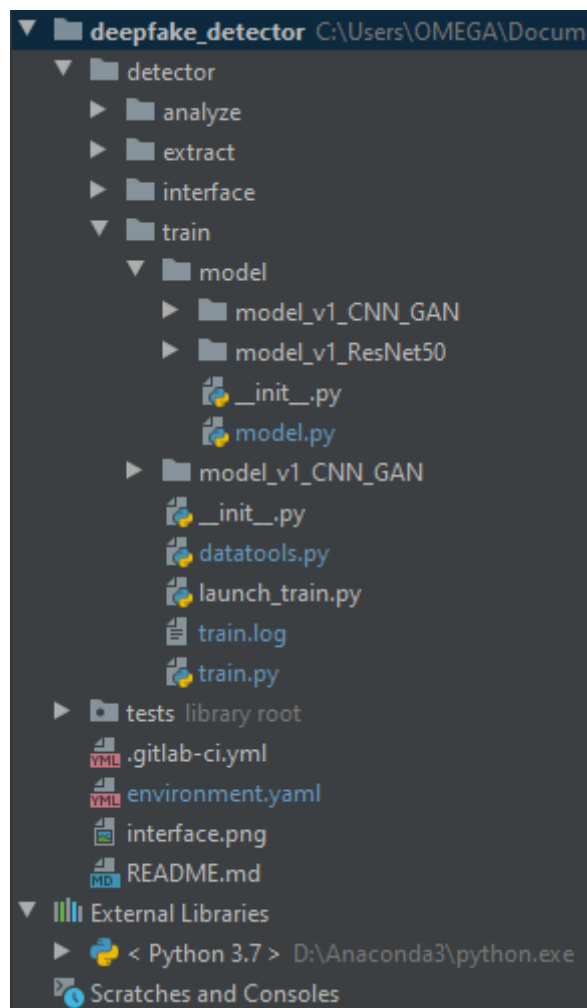


Figure 6 - Hiérarchie des fichiers du projet

Le répertoire « deepfake\_detector » est divisé en deux parties :

- detector (code du logiciel)
- tests (tests du code logiciel)



## Fichiers temporaires

Un ensemble composé de 3 fichiers distincts situés dans le package « extract » permet de stocker sous les différentes formes nécessaire le fichier d'entrée :

- images (image(s) du contenu)
- align-faces (visage(s) alignés du contenu)
- faces (visage(s) du contenu)

Les fichiers sont automatiquement détruits au début du lancement de l'extraction si ceux-ci existent.

## Tester le code logiciel

Des tests unitaires et implémentations sont présents dans le package « tests ».

Commande d'exécution des tests : `pytest nom_du_fichier.py`

## Script de pré-traitement de la donnée

Le script de pré-traitement `datatools.py` permet de pouvoir appliquer les différentes opérations nécessaire sur les différents jeux de données avant l'utilisation dans le script de gestion d'entraînement et test.

Il existe deux modes d'utilisations du script :

1. Extraction des images et visages venant d'images ou vidéos.
2. Extraction des matrices de co-occurrence des images

Lorsque l'extraction est terminée, il reste à répartir manuellement les images / .npz dans les différents répertoires (entraînement, validation, test).

**L'utilisation du script se fait uniquement par argument.**

*-i : Emplacement des fichiers à extraire*

*-o : Emplacement de sortie des fichiers temporaires ou .npz*

*-f : Emplacement des fichiers contenant uniquement les visages*

*-c : Mode (1 = option 1, 2 = option 2)*

### Exemple de commande avec l'option 1 :

```
-i D:\Dataset\images -f D:\Dataset\faces -o D:\Dataset\temp
```

### Exemple de commande avec l'option 2 :

```
-i D:\Dataset\deepfake -i D:\Dataset\real_face -o D:\Dataset\dataset
```

*Attention, le respect de l'ordre -i deepfake -i real\_face est impératif lors de la génération du jeux de données.*

## Script de gestion des entraînements/tests des modèles

Le script de gestion des entraînements/tests `launch_train.py` permet d'interagir avec les différentes fonctionnalité de la classe Train.

Il existe deux modes d'utilisations du script :

1. Entraînement, validation et test du modèle deep learning sélectionné
2. Test du modèle de deep learning **pré-entraîné** sélectionné

L'utilisation du script se fait uniquement par argument.

- i : Emplacement des fichiers d'entraînement du modèle
- v : Emplacement des fichiers de validation du modèle
- t : Emplacement des fichiers de test du modèle
- c : Modèle sélectionné (1 et 3 implémentés)
- o : Mode (0 = entraînement, validation, test, 1 = test uniquement)
- b : taille du batch pour l'ensemble des bases
- e : nombre d'épochs
- m : Emplacement du modèle deep learning pré-entraîné

#### Exemple de commande avec l'option 1 :

```
-i D:\Dataset\train -v D:\Dataset\valid -t D:\Dataset\test -c 3 -o 0  
-b 40 -e 50
```

Attention, le respect de l'ordre -i, -v, -t est impératif.

#### Exemple de commande avec l'option 2 :

```
-m D:\modele\modele_ResNet50 -t D:\Dataset\test -c 3 -o 1 -b 40 -e  
50
```

## Ajouter un modèle d'apprentissage profond

Les modèles pré-entraîné existant sont situés dans le dossier model

La classe Model contient l'ensemble des configurations pour chaque modèle.

1. Ajouter le nom du modèle dans la variable *models*

```
@staticmethod
```

```
def available_models(choice: int):
```

```
    """
```

```
    Display the different deep learning model implemented
```

```
    :param choice: the chosen model name
```

```
    :return: the model name
```

```
    """
```

```
    models = ['ResNet50', 'LSTM', 'CNN_GAN']
```

```
    return models[choice - 1]
```

2. Ajouter dans le constructeur *Model* la configuration du modèle (exemple ci-dessous)

```
if self.name == Model.available_models(1):
    try:
        base_model = ResNet50(include_top=False,
                                input_shape=(224, 224, 3), weights='imagenet')
        flatten_layer = Flatten() # block of features found on the image
        prediction_layer = tf.keras.layers.Dense(1, activation='sigmoid')
        self.optimizer =
        tf.keras.optimizers.SGD(tf.keras.optimizers.schedules.ExponentialDecay(
            initial_learning_rate=0.001, decay_steps=1000, decay_rate=0.95,
            staircase=False))

        ...
```

3. Ajouter un chargement de donnée spécifique dans *load\_dataset()*

Si le nouveau modèle nécessite un chargement de données non implémenté, il est possible d'ajouter sa propre configuration de chargement. Sinon il suffit de signaler celui correspondant en rajoutant la condition `self.model.name == Model.available_models(nombre)` pour celui correspondant (exemple ci-dessous).

```
if self.model.name == Model.available_models(1) or
self.model.name == Model.available_models(2):
    if self.model.name == Model.available_models(1):
        target_size = (224, 224)
    elif self.model.name == Model.available_models(2):
        target_size = (1, 299, 299)
    else:
        print('An error occurred when choosing target
size.')
```

```
        raise
    if mode == 0:
        train_image_generator =
        ImageDataGenerator(rescale=1. / 255)
        validation_image_generator =
        ImageDataGenerator(rescale=1. / 255)
    self.train_dataset =
    train_image_generator.flow_from_directory(batch_size=self.model.batch_size,
```

```
h_size, directory=self.input_content[0], shuffle=is_shuffle,  
target_size=target_size, class_mode='binary' )
```

...

#### 4. Lancer l'entraînement / test du modèle

Lorsque l'ensemble des étapes précédentes ont été réalisées dans le cadre de l'ajout d'un modèle, il suffit de paramétrer la commande en argument ([exemple de commande](#)).

N°	Nom de la méthode
1	test_should_compute_analyze
2	test_analyze_with_gan_model_detector
3	test_analyze_with_auto_encoder_model_detector
4	test_should_launch_analyze
5	test_constructor_extract
6	test_should_extract_content
7	test_should_crop_face
8	test_should_align_face
9	test_should_blur_face
10	test_should_calculate_co_occurrence_matrix
11	test_should_init_model
15	test_should_get_data_from_filename
16	test_on_filename_changed_disable
17	test_on_filename_changed_enable
18	test_on_button_contain_right_extension
19	test_on_quit
20	test_should_launch_auto_detector_and_gan_detector_then_compute_analyze
21	test_should_launch_extract_then_launch_analyze
22	test_should_extract_faces_and_crop_faces_and_align_faces_and_align_from_video
23	test_should_extract_faces_and_crop_faces_and_align_faces_and_align_from_image
25	test_should_launch_interface_and_browse_button_then_launch_detector
26	test_should_stop_extraction
27	test_should_stop_analyse
28	test_should_extract_and_analyse_image
29	test_should_extract_and_analyse_video
	*Non réalisable dans un délai de temps de réalisation satisfaisant
12	test_should_load_dataset*
13	test_should_train_model*
14	test_should_evaluate_model*
24	test_load_dataset_and_launch_train*

Action
Calculer la différence entre les prédictions des modèles de detections (auto-encodeur and GAN)
Prédit le contenu de avec le detecteur de DeepFake créé par GAN
Prédit le contenu de avec le detecteur de DeepFake créé par auto-encodeur
Lance l'analyse du contenu avec le controlleur
Initialise le constructeur Extract
Le contenu est extrait sous forme d'images et enregistré
Le contenu est redimensionné au niveau du ROI et enregistré
Le contenu est être alignée et enregistré
Le contenu est flouté
Le contenu est extrait sous forme de matrices de co-occurrence
Le modèle est initialisé correctement avec les paramètres correspondant
Le contenu du fichier compressé .npz est extrait
Le nom du fichier rentré est incorrecte
Le nom du fichier rentré est correcte
Le contenu en entrée est avec la bonne extension
L'interface reçoit l'ordre de se fermer
Le détecteur auto-encodeur et GAN analysent le contenu puis comparaison entre les résultats des deux
L'extraction du contenu et l'analyse du contenu sont lancées
Les images de la vidéo sont extrait, croppées et alignées
L'image est extraite, croppée et alignée
L'interface est lancée, le contenu est sélectionné et l'analyse est lancée
Le fichier est en cours d'extraction par le logiciel et interrompu par l'utilisateur
Le fichier a été extrait par le logiciel et est en cours d'analyse, l'utilisateur interrompt l'analyse
L'image est extraite et analysée par le logiciel
La vidéo est extraite et analysée par le logiciel
Le jeux de données est chargé en mémoire et divisé en plusieurs bases (entraînement, validation, test)
Le modèle sélectionné est entraîné avec le jeux de données en mémoire
Le modèle sélectionné est testé avec le jeux de données en mémoire
Le jeux de données est chargé en mémoire et l'entraînement est lancée

Attendu
Doit renvoyer la prédiction venant de <code>analyze_with_gan_model_detector</code> et <code>analyze_with_auto_encoder_model_detector</code> ainsi que les artefacts et l'étiquette
Retourne la prédiction, l'étiquette et les artefacts de l'image
Retourne la prédiction, l'étiquette et les artefacts de l'image
L'analyse doit se compléter en retournant la prédiction, l'étiquette et les artefacts de l'image
Le constructeur est correctement construit
Le fichier image existe et n'est pas vide
Le fichier faces existe, non vide et la ROI est retourné
Le fichier align_face existe et non vide
L'image retournée est différente de celle en entrée
Le contenu retourné est une matrice de forme 4*256*256*3
Le modèle possède des attributs aux valeurs conforme aux entrées
Le contenu du fichier et son étiquette sont retournés
Le nom du fichier détecté est incorrecte et le bouton de lancement de l'analyse est désactivé
Le nom du fichier détecté est correcte et le bouton de lancement de l'analyse est activé
L'extension du fichier est reconnu et l'analyse est exécutée
L'interface se ferme correctement
Les deux analyses sont effectuées, confiance supérieur à zéro et artefacts existant
Les fichiers temporaires existent et non vide, la confiance de l'analyse est supérieur à zéro et artefacts existant
Les fichiers temporaires existent, leur nombre de contenus supérieur à un et différentes entre les fichiers
Les fichiers temporaires existent, leur nombre de contenus est égale à un et différent entre les fichiers
L'interface est affichée et la barre de progression est correctement affichée, gérée et supprimée
La barre de progression disparaît et l'extraction est stoppée
La barre de progression disparaît et l'analyse est stoppée
L'interface affiche l'étiquette, la confiance dans la détection et les artefacts de l'image
L'interface affiche l'étiquette, la confiance dans la détection et les artefacts de la vidéo
Le jeux de données est correctement chargé en mémoire et correctement divisé en trois bases
Le modèle est entraîné et les métriques sont cohérentes
Le modèle est testé et les métriques sont cohérentes
Le jeux de données est correctement chargé en mémoire, correctement divisé en trois bases et le modèle est entraîné et les métriques sont cohérentes

[illegible]



<i>test unitaire (code couleur)</i>
<i>test d'intégration (code couleur)</i>



# Comptes rendus hebdomadaires

## **Compte rendu n°1 du 16/09/2019**

La première semaine consista à prendre en main les divers logiciels de création de "DeepFake" (FaceSwap, Face2Face, ...) recommandés par le tuteur pédagogique et dans le but d'essayer de comprendre le fonctionnement, l'installation de divers bibliothèques requises pour faire du deep learning (TensorFlow, Keras, CuDNN). Un essai a aussi été réalisé sur un ordinateur de l'école afin de pouvoir vérifier et utiliser la puissance de calcul du GPU disponible pour la génération de "DeepFake", hélas non utilisable car l'accès depuis une VM au matériel graphique est impossible. J'ai aussi pu me rendre compte de l'utilisation obligatoire, à terme d'un GPU puissant afin de pouvoir correctement générer (plusieurs milliers d'images) et entraîner les modèles sur de larges données ("DeepFake" ou non) de bonne qualité ainsi que de réduire leur temps de création.

## **Compte rendu n°2 du 23/09/2019**

La deuxième semaine consista à valider l'utilisation correcte des différents logiciels de création de "DeepFake" sur un ordinateur personnel, rédiger les manuels d'utilisation des logiciels, se former sur divers tutoriels des bibliothèques d'IA et débiter la rédaction de l'état de l'art sur les méthodes de détection de "DeepFake" existantes.

## **Compte rendu n°3 du 30/09/2019**

La troisième semaine consista à commencer à réfléchir à la rédaction du cahier des spécifications en vue de préparer la réunion la semaine suivante avec le tuteur pédagogique pour fixer un cahier des charges. J'ai aussi commencé à réfléchir à la rédaction du rapport final tout en continuant par ailleurs à me former sur les différentes technologies de deep learning et méthodes existantes.

## **Compte rendu n°4 du 07/10/2019**

La quatrième semaine consista après réunion avec le tuteur pédagogique et validation du cahier des charges, à rédiger le cahier des spécifications en accord avec les besoins du client. J'ai également réalisé la maquette de l'interface utilisateur. Enfin, j'ai fait un diagramme de Gantt prévisionnel par rapport au temps nécessaire pour l'analyse et l'implémentation des besoins client.

**Compte rendu n°5 du 14/10/2019**

La cinquième semaine consista à créer les principaux diagrammes statique (composant, objet, cas d'utilisation), continuer la rédaction du cahier des spécifications et à réfléchir sur le diagramme de classes.

**Compte rendu n°6 du 21/10/2019**

La sixième semaine consista à envoyer en validation le cahier des spécifications au client, faire le choix des méthodes à implémenter par rapport à l'étude faite sur l'état de l'art, reprendre la rédaction du rapport final et mettre à jour du diagramme de Gantt.

**Compte rendu n°7 du 04/11/2019**

La septième semaine consista à continuer la rédaction du rapport final, créer les diagrammes dynamique (activité et séquence) et le diagramme de classes.

**Compte rendu n°8 du 11/11/2019**

La huitième semaine consista à continuer la rédaction du rapport final, prendre en compte les remarques du tuteur pédagogique lors de la validation du cahier des spécifications et mettre à jour le Gantt. Suite à cela une nouvelle version du cahier des spécifications a été produite avec les corrections demandées et l'ajout de deux fonctionnalité omise qui a ensuite été soumise à validation et validé.

**Compte rendu n°9 du 18/11/2019**

La neuvième semaine consista à terminer le rapport final et commencer la création de la présentation de la soutenance du S9.

**Compte rendu n°10 du 25/11/2019**

La dixième semaine consista à finir la présentation de la soutenance du S9.

**Compte rendu n°11 du 02/12/2019**

La onzième semaine consista à déposer les différents documents demandés et à préparer la mise en place du développement logiciel.

**Compte rendu n°12 du 09/12/2019**

La douzième semaine consista à présenter le travail effectué durant la soutenance du S9, mettre en place le développement notamment de l'interface et télécharger/créer les différents jeux de données nécessaires

**Compte rendu n°13 du 16/12/2019**

La treizième semaine consista à terminer le développement de l'interface, faire les tests unitaires et commencer le développement du module Extraction

**Compte rendu n°14 du 06/01/2020**

La quatorzième semaine consista à terminer les téléchargements des jeux de données nécessaires, avoir une réunion avec le tuteur pédagogique sur l'avancé du projet et à implémenter les modèles CNN.

**Compte rendu n°15 du 13/01/2020**

La quinzième semaine consista à continuer le développement du module Extraction et tester l'implémentation des différents modèles CNN.

**Compte rendu n°16 du 20/01/2020**

La seizième semaine consista à terminer le module Extraction et constater les premiers résultats des implémentations des modèles avec l'apparition de problèmes associés.

**Compte rendu n°17 du 27/01/2020**

La dix-septième semaine consista à prendre rendez-vous avec le tuteur pédagogique afin de trouver des solutions aux problèmes constatés afin de continuer le développement des modèles.

**Compte rendu n°18 du 03/02/2020**

La dix-huitième semaine consista à continuer les tests de l'implémentation des différents modèles et constater les résultats.

**Compte rendu n°19 du 10/02/2020**

La dix-neuvième semaine consista à continuer les tests de l'implémentation des différents modèles et de commencer le développement de la visualisation des couches du réseau de neurones.

**Compte rendu n°20 du 17/02/2020**

La vingtième semaine consista à continuer les tests de l'implémentation des différents modèles et de commencer la rédaction des parties manquantes du rapport du projet de recherche.

**Compte rendu n°21 du 02/03/2020**

La vingt-et-unième semaine consista à continuer la rédaction du rapport, terminer les tests de l'implémentation des modèles et rédiger les différents guides d'utilisations tout en commençant la création du PowerPoint de présentation.

**Compte rendu n°22 du 09/03/2020**

La vingt-deuxième semaine consista à rencontrer le tuteur pédagogique / client pour faire le point sur les avancés du projet et à implémenter le réseau de neurones CNN-LSTM.

**Compte rendu n°23 du 16/03/2020**

La vingt-troisième semaine consista à continuer la création du PowerPoint pour la soutenance du S10 et à terminer la rédaction du rapport du projet de recherche dû à l'entraînement du modèle CNN-LSTM qui n'a pas fonctionné.



# Webographie

- [WWW1] *Auto-encodeur pour la compréhension de documents parlés*. Conférence. URL : <https://jep-taln2016.limsi.fr/actes/Actes%20JTR-2016/Papers/J14.pdf> (visité le 2019).

# Bibliographie

- [1] Ross Girshick ABHINAV SHRIVASTAVA Abhinav Gupta. « Training Region-based Object Detectors with Online Hard Example Mining ». Computer Science. Carnegie Mellon University, Facebook AI Research, 2016.
- [2] Ramprasaath R. Selvaraju Michael Cogswell Abhishek Das Ramakrishna Vedantam Devi Parikh Dhruv BATRA. « Grad-CAM : Visual Explanations from Deep Networks via Gradient-based Localization ». In : *IJCV* abs/1610.02391 (2019).
- [3] Vincent Vanhoucke Sergey Ioffe Jon Shlens Zbigniew Wojna CHRISTIAN SZEGEDY. « Rethinking the Inception Architecture for Computer Vision ». Computer Science. University College London, London, UK, 2016.
- [4] Edward J. Delp DAVUD GUERA. « Deepfake Video Detection Using Recurrent Neural Networks ». Computer Science. Purdue University, 2018.
- [5] Brian Dolhansky Russ Howes Ben Pflaum Nicole Baram Cristian Canton FERRER. « The Deepfake Detection Challenge (DFDC) Preview Dataset ». In : *Facebook AI* abs/1910.08854v2 (2019).
- [6] Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio IAN J. GOODFELLOW Jean Pouget-Adabie Medhi Mirza. « Generative Adversarial Nets ». Computer Science. Université de Montréal, Québec, Canada, 2014.
- [7] Matthias Nießner JUSTUS THIES Michael Zollhöfer. « Deferred Neural Rendering : Image Synthesis using Neural Textures ». Computer Science. Technical University of Munich, Stanford University, 2019.
- [8] Tero KARRAS, Samuli LAINE, Miika AITTALA, Janne HELLSTEN, Jaakko LEHTINEN et Timo AILA. « Analyzing and Improving the Image Quality of StyleGAN ». In : *CoRR* abs/1912.04958 (2019).
- [9] Pavel KORSHUNOV et Sébastien MARCEL. « DeepFakes : a New Threat to Face Recognition? Assessment and Detection ». Computer Science. Air Force Research Laboratory (AFRL) et the Defense Advanced Research Projects Agency (DARPA), 2018.
- [10] Pavel KORSHUNOV et Sébastien MARCEL. « Speaker Inconsistency Detection in Tampered Video ». Computer Science. Air Force Research Laboratory (AFRL) et the Defense Advanced Research Projects Agency (DARPA), 2018.

- [11] Tajuddin Manhar Mohammed Shivkumar Chandrasekaran Arjuna Flenner Jawadul H. Bappy JD.com Amit K. Roy-Chowdhury B. S. Manjunath LAKSHMANAN NATARAJ. « Detecting GAN generated Fake Images using Co-occurrence Matrices ». Computer Science. Santa Barbara, California, USA, Naval Air Warfare Center Weapons Division, China Lake, California, USA, University of California, Riverside, California, USA, 2019.
- [12] Andreas RÖSSLER, Davide COZZOLINO, Luisa VERDOLIVA, Christian RIESS, Justus THIES et Matthias NIESSNER. « FaceForensics++ : Learning to Detect Manipulated Facial Images ». In : *International Conference on Computer Vision (ICCV)*. 2019.
- [13] C. SANDERSON et B.C. LOVELL. « Multi-Region Probabilistic Histograms for Robust and Scalable Identity Inference. » In : t. 5558. LNCS, 2009, p. 199-208.
- [14] Kaiming He Xiangyu Zhang Shaoqing Ren Jian SUN. « Deep Residual Learning for Image Recognition ». In : *Microsoft Research arXiv/1512.03385v1* (2015).
- [15] Yuezun Li XIN YANG et Siwei LYU. « EXPOSING DEEP FAKES USING INCONSISTENT HEAD POSES ». Computer Science. University at Albany, State University of New York, USA, 2018.
- [16] Ming-Ching Chang YUEZUN LI et Siwei LYU. « In Ictu Oculi : Exposing AI Generated Fake Face Videos by Detecting Eye Blinking ». Computer Science. University at Albany, SUNY, 2018.
- [17] Siwei Lyu YUEZUN LI. « Exposing DeepFake Videos By Detecting Face Warping Artifacts ». Computer Science. University at Albany, State University of New York, USA, 2019.

# Glossaire

**réseau antagoniste génératif** Aussi appelé Generative adversarial networks (GAN), est un type de réseau de neurones qui par la mise en pair de réseaux dans une compétition permet avec un jeu d'entraide (théorie des jeux) la génération d'images à fort réalisme. [1](#)

**auto-encodeur** Type de réseau de neurones artificiels utilisé pour l'apprentissage non supervisé de caractéristiques discriminantes généralement dans le but de reproduire le plus fidèlement une même image. [1](#), [3](#), [8](#), [70](#), [71](#)

**batch size** Ce terme désigne la taille (lot) de l'échantillon du jeu de données envoyé au modèle lors de l'entraînement qui à la fin de son traitement met à jour ses prédictions et répète ce processus jusqu'à ce que tout le jeu de données soit parcouru. [19](#)

**CUDA** Compute Unified Device Architecture est une technologie propriétaire de NVIDIA qui permet d'utiliser le GPU afin d'exécuter en parallèle une série d'opération de calcul. [26](#), [33](#)

**deep learning** Aussi appelé apprentissage profond et qui consiste à modéliser avec un haut niveau d'abstraction des données grâce à des méthodes d'apprentissage automatique. [72](#)

**DeepFake** Technique de synthèse d'images basée sur l'intelligence artificielle permettant de remplacer, supprimer ou modifier le contenu dans un but de falsification. [72](#)

**epoch** Représente le nombre de fois que le modèle devra itérer dans le jeu de données selon la taille de l'échantillon (batch size) lors de l'entraînement. Son nombre est donc compris entre 1 à N le nombre d'itérations possible entre la taille du jeu de données et celui de l'échantillon. [10](#), [19](#)

**features** Fait référence aux descripteurs d'une image et sur lesquels l'analyse s'appuie afin de pouvoir faire une interprétation de ceux-ci. [9](#)

**GPU** Graphics Processing Unit (ou en français processeur graphique) permettant d'assurer les fonctions de calcul de l'affichage. [vi](#), [26](#), [33](#)

**MOA** Maître d'ouvrage, correspond au « Product Owner », ou autrement appelé commanditaire du projet. [23](#)

**modèle** Fichier contenant le résultat de l'entraînement du réseau de neurones. [4](#)

**réseaux de neurones** Système dont la conception est issu du fonctionnement du cerveau biologique (neurones). [3](#)





# Logiciel fakenews : détecteur d'images et de vidéos contrefaites

Alexandre Burnier-Framboret

Encadrement : Donatello Conte

## Objectifs

Les objectifs de ce projet :

- Détecter les visages d'individus falsifiés par auto-encodeur
- Détecter les visages d'individus falsifiés par réseau antagoniste génératif
- Afficher les artefacts qui ont permis de conclure sur la falsification



Visage artificiel créé par un réseau antagoniste génératif (StyleGAN2)

## Mise en œuvre

La mise en œuvre consiste à utiliser les méthodes du domaine de l'IA et plus précisément celles venant du deep learning qui ont déjà démontrées leur efficacité dans le milieu de la recherche et dont leur appropriation permettra la création d'un logiciel de détection de DeepFake pertinent.

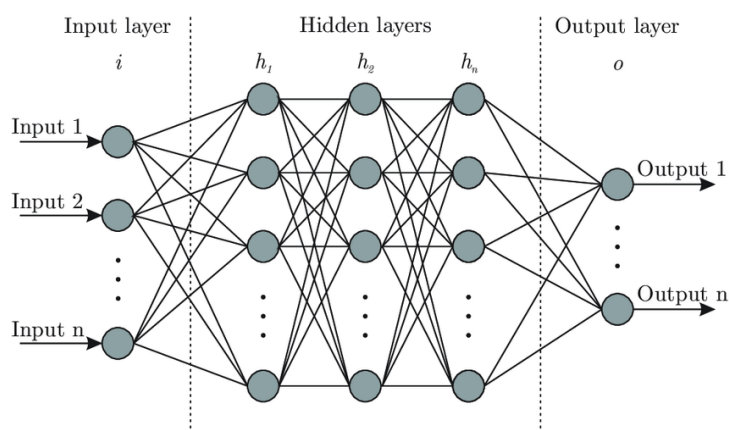
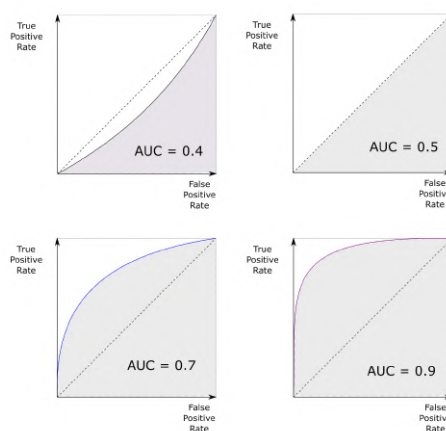


Schéma de la structure globale d'un réseau de neurones

## Résultats attendus

- Avoir un indicateur sur la falsification avec une confiance d'au moins 90%
- Pouvoir comprendre les éléments qui ont permis la décision des modèles de deep learning



Indicateur statistique de performance d'un modèle d'apprentissage profond

# Logiciel fakenews : détecteur d'images et de vidéos contrefaites

Alexandre Burnier-Framboret

Encadrement : Donatello Conte

## Objectifs

Les objectifs de ce projet :

- Détecter les visages d'individus falsifiés par auto-encodeur
- Détecter les visages d'individus falsifiés par réseau antagoniste génératif
- Afficher les artefacts qui ont permis de conclure sur la falsification



Visage artificiel créé par un réseau antagoniste génératif (StyleGAN2)

## Mise en œuvre

La mise en œuvre consiste à utiliser les méthodes du domaine de l'IA et plus précisément celles venant du deep learning qui ont déjà démontrées leur efficacité dans le milieu de la recherche et dont leur appropriation permettra la création d'un logiciel de détection de DeepFake pertinent.

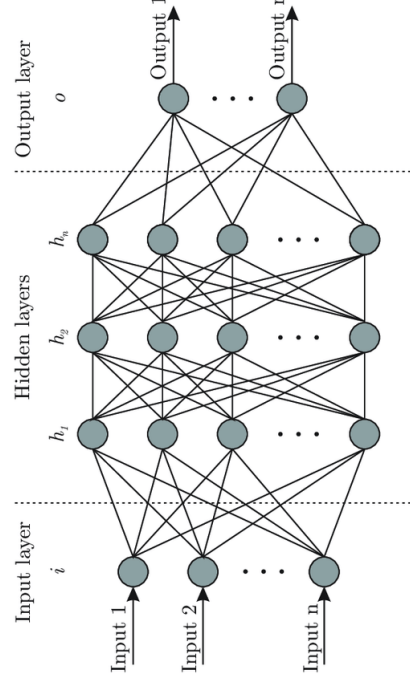
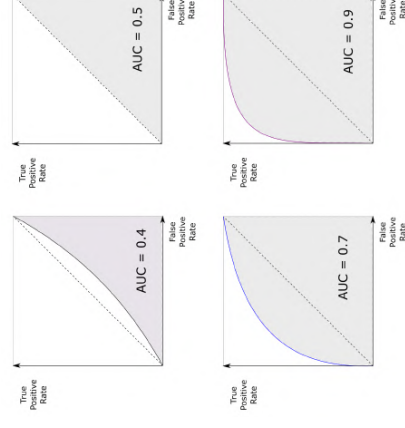


Schéma de la structure globale d'un réseau de neurones

## Résultats attendus

- Avoir un indicateur sur la falsification avec une confiance d'au moins 90%
- Pouvoir comprendre les éléments qui ont permis la décision des modèles de deep learning



Indicateur statistique de performance d'un modèle d'apprentissage profond

# Logiciel fakenews : détecteur d'images et de vidéos contrefaites

## Résumé

Apparu initialement de manière fictive en 2008, le phénomène du "DeepFake" s'est accéléré à partir de 2016, avec l'apparition de logiciels libres permettant sa réalisation sans connaissance particulière. Le "DeepFake" (mot valise entre deep learning et fake) consiste généralement à partir d'une image ou vidéo à substituer par des méthodes d'apprentissage profond le visage de la personne-cible par celui d'un autre individu existant ou créé artificiellement. Ce phénomène est alors particulièrement inquiétant au niveau sociétal par ses possibles effets. Ainsi, de nombreux acteurs travaillent activement sur ce sujet afin de détecter au plus tôt et précisément ces contenus dans le but d'empêcher leurs potentiels effets sur la société tels que l'apparition de "FakeNews" ou autres. Ce projet de recherche et développement porte donc sur l'identification des images et vidéos "DeepFake" grâce à l'utilisation de méthodes de **deep learning**.

## Mots-clés

intelligence artificielle, apprentissage profond, reconnaissance de formes, classification

## Abstract

Initially appeared in a fictitious way in 2008, the phenomenon of "Deepfake" accelerated from 2016, with the appearance of free software allowing its realization without particular knowledge. The "Deepfake" (word suitcase between deep learning and fake) usually consists of an image or video to substitute by methods of deep learning the face of the person-target by another artificially created or existing individual. This phenomenon is particularly worrying at the societal level because of its possible effects. Thus, many actors are actively working on it in order to detect as soon as possible and precisely these contents in order to prevent their potential effects on society such as the appearance of "Fakenews" or others. This research and development project therefore focuses on the identification of "Deepfake" images and videos through the use of deep learning methods.

## Keywords

artificial intelligence, deep learning, pattern recognition, classification