SYSTÈMES INTELLINGENTS AVANCÉS

COMPTE RENDU DE TP

Auteurs :

EL ALOUANI Naofel

COUTAREL Allan

ESIREM 5A INFOTRONIQUE – ILC GROUPE TP2

Année 2023/2024

Table des matières

I. Liste des figures

II. Acronymes

CNN	Convolutional Neural Network
CPU	Central processing unit
DL	Deep Learning
DPM	Deformable Part Model
DTO	Data Transfer Object
FPS	Frame Per Second
Go	Giga-octet
GPU	Graphics Processing Unit
IDE	Environnement de Développement
Ms	Millisecondes
ONNX	Open Neural Network Exchange
R-CNN	Region based Convolutional Neural Network
UML	Unified Modeling Language
VS Code	Visual Studio Code
YOLO	You Only Look Once

III. Introduction Générale

Nous vivons dans une ère où le traitement d'image et la reconnaissance faciale sont au cœur de nombreux domaines tels que la sécurité, la santé, le marketing ou encore le divertissement. L'enregistrement de vidéo est de plus en plus fréquent grâce aux nouvelles technologies intégrées dans les smartphones.

Il suscite cependant de nouvelles questions concernant le droit à l'image. Par exemple, lors de reportage ou d'interview dans des lieux publics. D'un côté, il y a le besoin d'informer un auditoire ce qui est l'essence même de tous médias. De l'autre côté il y a la nécessité de respecter la vie privée des individus présents à leur insu sur une vidéo. Une problématique se pose donc : Comment le développement de telles technologies peut coexister avec les normes éthiques et légales ?

Notre projet permet d'apporter une solution à ce problème, permettant en 4 clics, d'éviter toutes poursuites liées au respect du droit à l'image d'une personne lors de la publication d'une photo ou d'une vidéo. Ce rapport de projet se structure en quatre chapitres distincts afin d'aborder cette question. Le premier chapitre détaille les spécificités du projet, incluant son architecture, l'algorithme développé, ainsi que les technologies impliquées. Le deuxième chapitre se concentre sur le détecteur de visages, explorant l'état de l'art des détecteurs d'objets et les processus d'entrainement et d'inférence du modèle. Le troisième chapitre traite de la comparaison des visages, examinant différentes techniques et outils couramment utilisés dans ce domaine. Enfin, le quatrième chapitre présente l'interface utilisateur, crucial pour l'interaction effective avec le programme, avant de conclure sur une synthèse générale qui reviendra sur les enjeux et les perspectives de ce projet ambitieux.

IV. Chapitre 1 : Présentation du projet

1. Introduction

À travers ce chapitre, nous explorerons la conception du projet. Afin de bien cibler les fonctionnalités et les besoins métiers du projet, nous entamerons notre exploration par une analyse détaillée des choix stratégiques en matière de solutions techniques, en mettant en lumière les raisons sous-jacentes à ces décisions et leur impact sur l'objectif principal du projet.

Par la suite, nous plongerons dans l'étude de l'architecture du projet qui a été soigneusement élaborée avant le commencement de la programmation. Cette section mettra l'accent sur l'importance d'une architecture optimisée pour la réalisation d'un projet robuste et insensible à l'ajout de nouvelles fonctionnalités. Nous étudierons les technologies et outils utilisées afin de mener à bien le développement du projet.

2. Fonctionnalités

Le problème ayant donné naissance à ce projet est le respect du droit à l'image. En effet, il est primordial de respecter le droit à l'image d'individus présent lors d'un enregistrement vidéo dans un lieu public afin qu'ils ne soient pas filmés à leur insu.

Afin de garantir l'anonymat de chaque individu filmé à son insu, différentes solutions peuvent être mises en place. L'une des plus efficaces et des plus couramment utilisé est le floutage de visage. C'est la solution technique que nous avons choisi d'adopter au sein de notre projet afin de garantir l'anonymat de chaque passant d'une vidéo.

Néanmoins, le floutage d'un même visage sur chaque image d'une vidéo est un processus particulièrement long et non trivial pour un utilisateur. C'est pour cela que la seconde fonctionnalité clé de notre projet est la reconnaissance d'individu. En effet, notre projet a pour objectif de détecter chaque visage présent sur une vidéo et de les regrouper selon des individus. Cette solution permettra donc à un utilisateur :

i. D'envoyer une vidéo qui sera analysée par le programme afin de détecter tous les visages et de les regrouper selon des individus.
ii. De sélectionner les individus à flouter
iii. De télécharger la vidéo avec les individus floutés

Le floutage du visage de chaque visage d'une même personne sera donc automatique et entièrement réalisé par notre programme. L'utilisateur n'aura qu'a upload sa vidéo et choisir les personnes à flouter.

3. Architecture du projet

Afin de garantir une robustesse et une évolutivité sans faille, l'architecture du projet a été élaboré avant même le début du développement. Cette décision stratégique est cruciale pour éviter la dispersion des efforts et assurer une focalisation sur les éléments essentiels. Elle contribue à une planification méticuleuse et à l'élaboration d'une infrastructure agile, prête à s'ajuster avec souplesse aux modifications et aux expansions futures. L'efficacité de l'architecture se reflète directement dans la robustesse et l'adaptabilité du développement face aux défis rencontrés. Comme le souligne l'expert en programmation Robert C. Martin :

> *"Good architecture makes the system easy to understand, easy to develop, easy to maintain, and easy to deploy. The ultimate goal is to minimize the lifetime cost of the system and to maximize programmer productivity."* [1]

C'est en suivant cette philosophie que nous avons conçu le programme en adoptant l'architecture du diagramme Unified Modeling Language (UML) présenté par la figure 1.

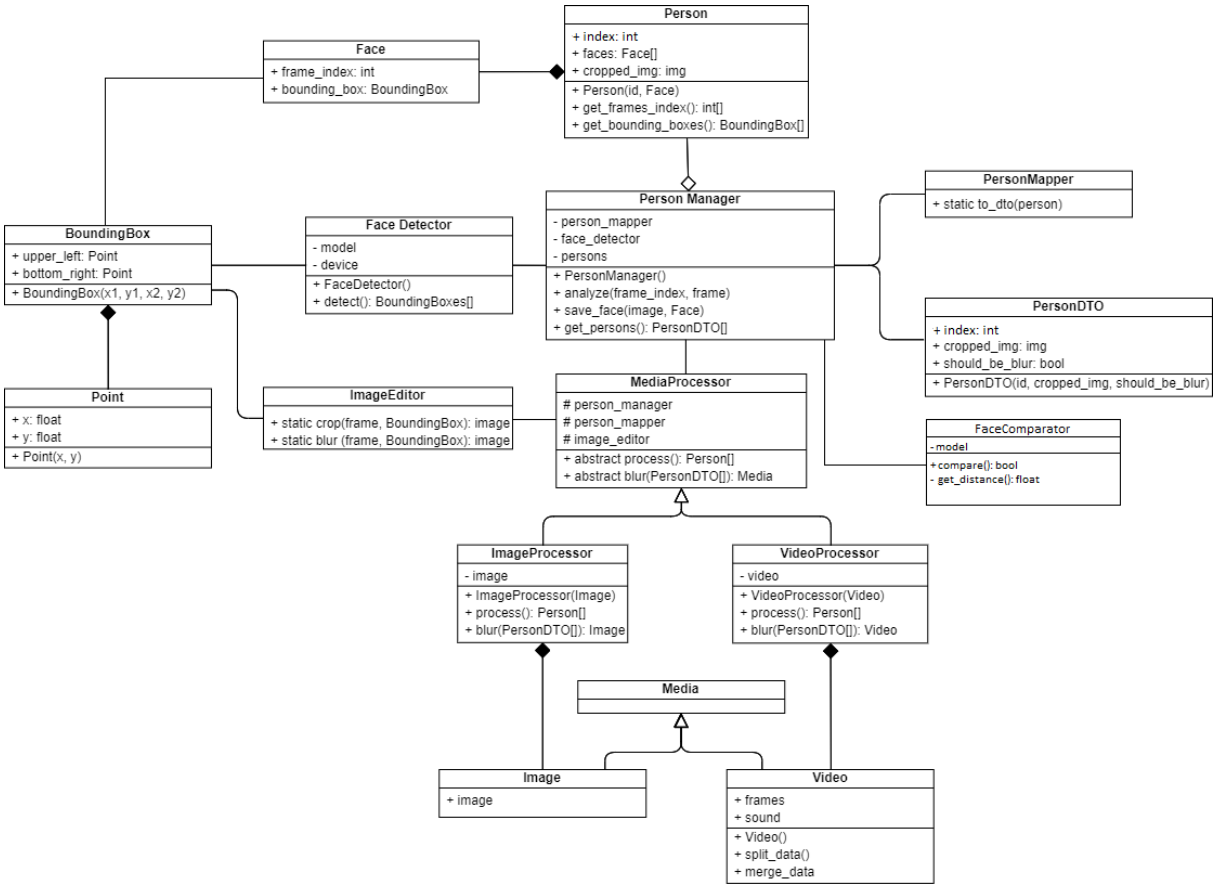

Figure 1 - Diagramme UML de l'architecture du projet

Cette architecture a été soigneusement conçu afin de faciliter le développement de ce logiciel tout en optimisant au maximum la mémoire. Certaines classes ont été légèrement modifié durant le développement du projet mais seront expliqués ci-dessous.

Voici une brève description des classes présentent sur la figure 1 :

- **BoundingBox** : Cette classe contient deux Points. Qui correspondent à des coordonnées. Un objet BoundingBox est créé lorsqu'un visage a été détecté. Les coordonnées de cette box indiquent les coordonnées du visage sur l'image.
- **Comparison** : La classe Comparison n'est pas présente sur le diagramme et joue un rôle d'interface. Un objet Comparison est retourné après la comparaison de deux visages. Elle contient une variable Boolean *is_same_person* ainsi qu'une variable *distance* indiquant la distance entre les deux visages.
- **Face** : La classe Face possède un index indiquant l'image à laquelle a été détecté le visage ainsi qu'un objet Prédiction qui sera détaillé plus tard.

- **FaceDetector** : Cette classe est chargée d'effectuer la détection des visages. Elle comprend une méthode *detect*, une méthode *warm_up* ainsi qu'un attribut *model* représentant le réseau de neurones responsable de la détection des visages.
- **FaceComparator** : La classe FaceComparator est une classe abstraite responsable de la comparaison des visages afin de savoir si deux visages représentent les mêmes personnes. Plusieurs classes héritaient de cette classe car nous avons implémenté plusieurs moyens de comparaison. Tout ceci sera détaillé dans le chapitre 3.
- **Image** : La classe Image hérite de la classe Média. Elle permet au logiciel de traiter également les images.
- **ImageEditor** : Cette classe est composée de méthodes statiques permettant du traitement d'image comme le découpage ou encore le foutage d'une zone définie par une BoundingBox.
- **Media** : La classe Média est abstraite et possède deux classes filles : la classe Image et la classe Video.
- **MediaProcessor** : MediaProcessor est également une classe abstraite. Elle est la classe mère de ImageProcessor et VideoProcessor.
- **Person** : La classe Person représente un individu. Un individu contient un id, une liste de Face, une *cropped_face* qui représente le visage de référence de cette personne. La *cropped_face* est utilisée lors des comparaisons afin de savoir si un visage appartient à une personne. Il contient également une *cropped_face_confidence* représentant la qualité de la *cropped_face*.
- **PersonDTO** : un Data Transfer Object (DTO) est un objet utilisé pour simplifier le transfert d'un objet. Il est utilisé ici pour envoyer au frontend les individus détectés sur la vidéo.
- **PersonManager** : La classe PersonManager est la classe clé du regroupement des visages selon des individus. Elle est responsable de l'attribution de visage à un individu.
- **PersonMapper** : Cette classe possède une méthode statique permettant de convertir un objet de type Person en PersonDTO.
- **Point** : La classe Point représente comme son nom l'indique un Point. Elle est utilisée dans la classe BoundingBox afin de définir le coin supérieur gauche et inferieur droit de la box.

- **Prediction** : La classe Prediction est utilisée en sortie d'une détection de visage. Elle contient une BoundingBox ainsi qu'un score de confiance utilisé afin de définir la qualité du visage détecté.
- **Video** : La classe Video est une classe utilisée pour charger une vidéo. Elle contient le chemin vers la vidéo, le nombre d'images par seconde, l'audio de la vidéo ainsi que le nombre total d'images dans la vidéo. Cette classe contient également des méthodes permettant d'interagir avec la vidéo comme la récupération de la nième image ou encore l'extraction de la piste audio.
- **VideoProcessor** : Cette classe est responsable de l'analyse de la vidéo ainsi que de la correction et de l'enregistrement de la vidéo de sortie.

4. Technologies utilisées (Git pour le versioning etc…)

Pour le développement di projet, nous avons choisi Visual Studio Code (VS Code) en tant environnement de développement (IDE). VS Code est un éditeur de code libre développé par Microsoft offrant une multitude d'extensions.

Afin de bien gérer les dépendances de notre projet nous avons mis en place un environnement virtuel Python grâce à venv. L'utilisation de venv était essentielle pour garantir que chaque membre de l'équipe travaille dans un environnement isolé et contrôlé, ce qui est primordial pour éviter des conflits de dépendances avec des projets préalablement menés.

Concernant le versioning, nous avons intégré Git dans notre organisation. Git est un outil incontournable pour le suivi des modifications, la collaboration et la gestion de version dans un projet. Il nous a permis de maintenir un historique détaillé des changements, ce qui est vital pour comprendre l'évolution du projet.

Chacun de ces outils a été choisi non seulement pour ses fonctionnalités individuelles, mais aussi pour la façon dont ils se complètent et s'intègrent ensemble.

5. Conclusion

Ce chapitre nous a permis d'introduire les besoins du projet, ses fonctionnalités clés du projet et de présenter son architecture. Nous avons étudié les besoins liés au respect du droit à l'image et comment notre solution de floutage de visage, combinée à une reconnaissance d'individu sophistiquée, répond à ces besoins tout en facilitant la tâche pour l'utilisateur final. L'architecture du projet, conçue avec précaution, sert de fondement solide, assurant non

seulement la robustesse mais aussi la flexibilité pour accueillir les futures extensions et modifications.

Le chapitre 2 présentera l'étude et le développement du détecteur de visage qui joue un rôle majeur dans la bonne réalisation du projet.

V. Chapitre 2 : Détecteur de visages

1. Introduction

Le premier chapitre a permis d'introduire le besoin qui a donné naissance à ce projet ainsi que tous les détails liés à l'organisation de son développement.

Nous commencerons ce nouveau chapitre par un état de l'art des détecteurs d'objets dans le monde des réseaux de neurones afin d'analyser les solutions s'offrant à nous pour détecter des visages sur une image. Cet état de l'art sera suivi par l'entrainement du modèle choisi puis d'une explication de son implémentation au sein du projet.

2. État de l'art des détecteurs d'objets

A. Définition

Il est tout d'abord nécessaire de définir ce que sont les détecteurs d'objets et pourquoi leur rôle est crucial au sein de notre projet.

La détection d'objets en Deep Learning (DL) est un processus visant à détecter des classes visuelles dans une image. Les réseaux de neurones conçus pour la détection d'objets fournissent généralement les coordonnées d'une boite sur l'image contenant l'objet. Cette boite est souvent accompagnée d'un score de confiance indiquant la probabilité que l'objet détecté appartienne à la classe renvoyée.

L'objectif de la détection d'objet est d'avoir une précision la plus élevée possible tout en ayant un temps d'inférence le plus petit possible. Une inférence est le terme utilisé afin de décrire la réalisation d'une prédiction par un réseau de neurone.

La détection d'objets est utilisée aujourd'hui dans divers domaines tels que la sécurité, l'industrie, la robotique et bien d'autres encore.

B. Historique

La détection d'objets a énormément évolué ces 20 dernières années. Les premières approches de la détection d'objets. Son évolution est divisée en trois ères [2] :

- Avant 2014, L'ère de la détection d'objets traditionnelle

- Viola-Jones Detector – 2001 : ce détecteur est l'un des premiers détecteurs d'objets efficaces. Ce détecteur était principalement utilisé pour la détection de visages puis est devenu une base pour de nombreuses méthodes de détection.
- Deformable Part Model (DPM) – 2008 : ce modèle a introduit la régression de bounding box (ou boite englobante en français) qui est devenu par la suite un élément central de la détection d'objets.

- Après 2014, L'ère de la détection d'objets après la Deep Learning
  - Region based Convolutional Neural Network (R-CNN) – 2014 : cet algorithme a introduit l'utilisation de Convolutional Neural Network (CNN) pour la détection d'objets ce qui a permis de franchir une nouvelle marche en termes de précision.
  - Mask R-CNN – 2017 : ce modèle est une évolution de Faster R-CNN (2015) qui se distingue grâce à sa capacité à effectuer non seulement de la détection d'objets, mais également de la segmentation d'instances. La segmentation d'instance est un concept visant à découper chaque objet pixel par pixel contrairement à la détection d'objets qui définit une zone rectangulaire dans laquelle se trouve l'objet détecté.
- You Only Looke Once (YOLO) – 2016
  - Le modèle YOLO a marqué le début d'une nouvelle ère : comme son nom l'indique, le modèle analyse l'image une seule fois, le rendant très rapide et introduisant la détection en temps réel.

Comme nous avons pu le voir dans l'historique précédent, il existe deux catégories de méthodes de détection d'objets : les détections en deux étapes et en une étape. Ces détecteurs résolvent deux tâches indispensables pour la détection d'objets :

1. Trouver un nombre d'objets
2. Classer chaque objet et estimer sa taille

Les détecteurs en deux étapes (R-CNN par exemple) commencent à calculer les régions approximatives des bounding boxes grâces aux caractéristiques extraites de l'image puis les utilisent pour classifier l'objet et effectuer la régression de la bounding box. Leur architecture est donc composée de deux étapes : une première qui est chargée de définir approximativement la région d'un objet et la seconde qui classifie l'objet et affine la bounding box. Cette

architecture fait des détecteurs en deux étapes des détecteurs dotés d'une haute précision mais leur temps d'inférence est généralement plus long.

Les détecteurs en une étape tels que YOLO par exemple, prédisent des bounding boxes sans passer par l'étape de définition d'une région pour l'objet. Ce processus est donc bien plus rapide et permet à ces modèles d'effectuer de la détection d'objet en temps réel. Leur petit point faible est qu'ils ne sont pas très performants pour reconnaitre des objets de forme irrégulière contrairement aux détecteurs en deux étapes.

C. Aujourd'hui

Les modèles YOLO n'ont cessé d'évoluer jusqu'aujourd'hui. Le dernier modèle YOLO est YOLOv8, développé par les créateurs de YOLOv5. Ce modèle est open-source et peut effectuer quatre tâches différentes :

- La détection d'objets
- La segmentation d'instances
- La classification d'images
- L'estimation de pose, permet de déterminer la position et l'orientation des éléments d'un objet (souvent un humain)

YOLOv8 existe en 5 versions : n, s, m, l, x. Ces versions varient selon leur nombre de paramètres allant de quelques millions de paramètres pour la version YOLOv8-n à 70 millions pour la version YOLOv8-x. La comparaison des performances des version de YOLOv8 est représentée par la figure 2.

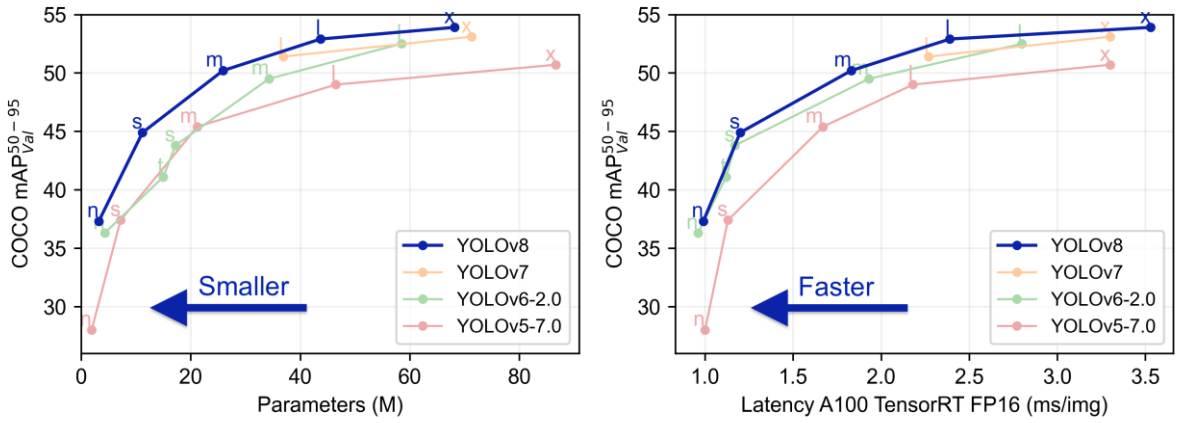

Figure 2 - Performances des versions de YOLOv8 [3]

On remarque que plus le modèle est léger, plus son temps d'inférence est faible atteignant une milliseconde (ms) pour la version n. En revanche, plus le modèle est lourd, plus il est précis.

Pour atteindre une détection de visages à la fois rapide et efficace, essentielle pour garantir une expérience utilisateur fluide et réactive, notre choix s'est porté sur YOLOv8-n. En optimisant le modèle pour la rapidité, nous visons à minimiser le délai entre la lecture d'une image et la détection de visages.

3. Entrainement du modèle

Les développeurs de YOLOv8 l'ont placé dans une librairie, créant tout un écosystème autour du modèle afin d'avoir une couche d'abstraction qui permet aux utilisateurs de prendre en main aisément ce réseau de neurones. Cette librairie est appelée *ultralytics* [3] et disponible gratuitement via GitHub ou l'utilitaire *pip*.

Cette librairie fournit une fonction permettant d'entrainer YOLOv8 sur un dataset personnalisé.

D. Choix du dataset

Afin d'avoir la détection de visages la plus précise possible, il est nécessaire de préparer un dataset complet sur lequel le modèle YOLOv8 pourra s'exercer. Après avoir réalisé quelques recherches sur les sites Kaggle et Roboflow qui sont des références universelles en termes de bibliothèque de dataset dans le monde, un dataset nous a paru idéal pour notre besoin. En effet, le dataset Face-Detection-Dataset [4] ne regroupe pas moins de 16.7k images de visages pour seulement 5 Giga-octets (Go). Chaque image est annotée et associée à un fichier texte contenant les cordonnées de la bounding box présente sur l'image.

Pour pouvoir servir de base d'entrainement d'un détecteur YOLO, le dataset nécessite une architecture spécifique nommée YOLO. Le dossier racine du dataset doit contenir deux dossiers principaux, "images" et "labels". Le dossier "images" contient toutes les images du dataset, tandis que le dossier "labels" contient les fichiers de labels associés aux images. À l'intérieur du dossier "images", les images sont généralement réparties dans des sous-dossiers distincts tels que "train", "val" et "test", correspondant respectivement aux données d'entraînement, de validation et de test. Chaque image du dossier "images" possède un fichier de label correspondant dans le dossier "labels" sous forme de fichier .txt portant le même nom que l'image. Ces fichiers de label contiennent les annotations d'objet au format YOLO,

spécifiant les coordonnées des bounding boxes et les classes des objets présents dans les images. En plus de ces dossiers, un fichier de configuration *dataset.yaml* est nécessaire afin de décrire les détails du dataset, y compris les chemins vers les dossiers d'images et de labels, ainsi que d'autres paramètres essentiels comme les classes d'objets utilisés ou le nombre total d'images.

Le dataset que nous avons utilisé n'étant pas au format YOLO, un petit script Python a permis de le mettre sous la bonne architecture.

```
Face-Detection-Dataset/
├── images/
│   ├── train/
│   ├── val/
│   └── test/
├── labels/
│   ├── train/
│   └── val/
└── dataset.yaml
```

Figure 3 - Architecture du dataset au format YOLO

E. Entrainement

Le dataset est désormais prêt, le modèle YOLOv8-n peut donc être entrainé dessus. Pour ce faire, la librairie *ultralytics* fournis une couche d'abstraction permettant de charger un modèle YOLOv8 très simplement et de l'entrainer avec seulement les 3 lignes suivantes :

```
from ultralytics import YOLO
model = YOLO('yolov8n.pt')
results = model.train(data='./Face-Detection-Dataset/dataset.yaml', epochs=10)
```

Nous avons choisi d'entrainer le modèle sur 100 epochs afin d'avoir le détecteur le plus performant possible en termes d'efficacité. Comme le montre la figure La Mean Average Precision (mAP) à un seuil Intersection over Union (IoU) de 0.5 est de 0.891 ce qui signifie que notre détecteur de visages basé sur YOLOv8n est très performant en conditions de test avec une efficacité de 89.1% de vrais positifs.

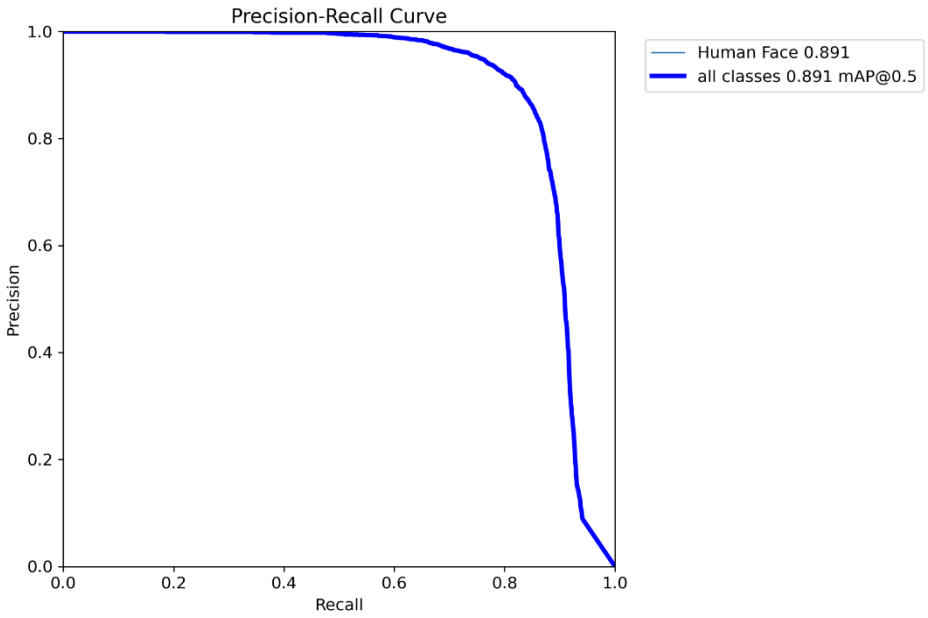

Figure 4 - Précision du détecteur de visages

4. Implémentation

L'implémentation du détecteur de visages repose sur une classe *FaceDetector* qui hérite de la classe abstraite *Model*. La classe abstraite *Model* définit un modèle générique avec une méthode abstraite *warm_up*, permettant ainsi d'assurer une initialisation préalable du modèle.

Dans le constructeur de la classe *FaceDetector*, le chemin du modèle YOLO spécifique pour la détection des visages que nous avons entrainé au préalable est fourni, et le modèle est initialisé en fonction de ce chemin. Le modèle est ensuite déplacé sur le GPU s'il est disponible, permettant ainsi une accélération matérielle pour les opérations de détection.

La méthode *warm_up* est implémentée pour « préchauffer » le modèle en effectuant une prédiction initiale avec une image vide. Cela permet de charger les poids du modèle et de préparer les ressources nécessaires pour les inférences ultérieures. Un message de journalisation indique que le modèle YOLO pour la détection de visages est prêt à l'utilisation après cette étape.

La méthode *detect* prend une image en entrée et effectue la détection de visages en utilisant le modèle YOLO. Les prédictions sont extraites des résultats de la détection, et une liste d'objets *Prediction* est renvoyée. Chaque objet *Prediction* encapsule les coordonnées d'une boîte englobante (bounding box) et la confiance associée à la détection du visage.

Cette implémentation offre une solution robuste pour la détection de visages en utilisant YOLO, tout en fournissant une structure modulaire qui peut être étendue pour intégrer d'autres fonctionnalités ou améliorations spécifiques aux besoins du projet.

5. Conclusion

Ce chapitre a exploré en détail le domaine des détecteurs d'objets, en mettant particulièrement l'accent sur la détection de visages, essentielle pour notre projet. Nous avons commencé par définir les détecteurs d'objets en Deep Learning (DL) et leur rôle crucial dans notre contexte. La détection d'objets consiste à identifier des classes visuelles dans une image, en fournissant les coordonnées d'une boîte contenant l'objet, accompagnée d'un score de confiance.

L'analyse historique a révélé l'évolution significative de la détection d'objets au cours des deux dernières décennies. Des méthodes traditionnelles telles que le détecteur Viola-Jones ont ouvert la voie, suivies par l'ère révolutionnaire de la détection d'objets après l'avènement du Deep Learning. Des modèles tels que R-CNN, YOLO, et Mask R-CNN ont marqué des étapes cruciales, aboutissant à la création de YOLOv8, notre choix pour la détection de visages.

Nous avons opté pour YOLOv8-n en raison de sa légèreté et de sa rapidité, essentielles pour garantir des performances en temps réel dans notre interface utilisateur. L'entraînement du modèle a été réalisé avec le dataset Face-Detection-Dataset, soigneusement préparé pour répondre aux exigences spécifiques de YOLO.

En somme, ce chapitre a jeté les bases de la détection de visages dans notre projet, fournissant un aperçu approfondi de l'état de l'art, de l'évolution historique, du choix du modèle, de l'entraînement, et des considérations liées au dataset. Ces éléments constituent la fondation sur laquelle repose notre capacité à intégrer efficacement la détection de visages dans notre application de floutage interactif pour des médias vidéo et des images.

VI. Chapitre 3 : Comparateur de visages

1. Introduction

Après avoir conçu avec succès un détecteur de visage performant à l'aide de YOLO, il était temps d'attaquer la partie la plus importante du projet : le regroupement des visages en différents individus. Ce domaine est connu sous le nom de *face recognition* en Deep Learning et est un domaine fascinant en pleine expansion.

Ce chapitre commencera donc par un état de l'art de la *face recognition* afin de comprendre ses utilisations, ses limites, ainsi que les projets publics permettant de déterminer si deux photos représentent la même personne. Par la suite, l'implémentation de la classe *FaceComparator* au sein du projet sera détaillée puis une dernière partie expliquera en détail l'algorithme de correction que nous avons développé afin d'avoir les résultats les plus fiables possibles.

2. État de l'art de la face recognition

A. Présentation

La face recognition est le fait de réussir à déterminer si deux visages sont les mêmes ou non. En Deep Learning, afin de comparer si deux visages appartiennent à la même personne, les images de visages sont passées dans un modèle qui va agir comme un extracteur de caractéristiques (ou *features extractor)*. Ce modèle est généralement un CNN car ce type de réseaux de neurones s'avère être très performant pour ce genre de tâches. Les extracteurs de caractéristiques prennent une image en entrée et retournent un vecteur de nombres représentant les caractéristiques de l'image entrée. Pour avoir un extracteur de caractéristique fonctionnel, le modèle doit être entrainé sur deux nombreuses images afin de cerner le mieux possible les caractéristiques de chaque image.

Une fois les caractéristiques extraites des deux images contenant les visages, une distance est calculée entre les deux vecteurs respectifs. Il existe deux méthodes couramment utilisées pour calculer la distance de manière efficace entre ces deux vecteurs :

- La distance euclidienne : Cette méthode mesure la distance dite géométrique entre deux points ou deux vecteurs. Plus la distance est courte plus la similarité entre les deux entrées est élevée.

- La distance cosinus : Cette méthode mesure l'angle cosinus entre les deux vecteurs d'entrée. Elle est très utilisée pour les vecteurs car elle est moins sensible à la magnitude de leurs données.

Enfin, lorsque la distance est calculée, elle est comparée à un seuil afin de savoir si les deux visages présents sur les deux images sont les mêmes. Si la distance est supérieure au seuil, les visages sont considérés comme différents. En revanche, si elle est inférieure au seuil, les deux visages sont considérés comme appartenant à la même personne [5].

B. Face-recognition

Durant la recherche de librairies et de modèles implémentant la face recognition pour déterminer si deux visages appartiennent à la même personne, la librairie Python *face_recognition* [6] est très vite ressortie comme une solution à notre problème pour extraire et comparer les caractéristiques des visages. Cette librairie est très complète et dispose de nombreuses fonctionnalités telles que :

- La détection de visages
- La localisation de caractéristiques d'un visage
- La classification de visage pour certaines célébrités
- La face recognition

Il a donc été décidé d'implémenter cette librairie au sein de notre classe *FaceComparator*. Après avoir réalisé quelques tests sur les performances de la librairie, deux problèmes se sont révélés :

- Les temps d'inférence extrêmement longs : environ 6 secondes pour extraire les caractéristiques de deux visages et les comparer. Étant donné le nombre de visages pouvant être présent sur une vidéo filmée au minimum en 30 Frame Per Second (FPS), il était inconcevable d'avoir un temps d'inférence aussi long.
- Des vecteurs de caractéristiques vides sur certains visages détectés par la classe *FaceDetector.*

Ces problèmes de performances nous ont conduit à continuer nos recherches afin de trouver un modèle permettant d'extraire rapidement les caractéristiques des visages.

C. DeepFace

Le premier comparateur a montré des résultats décevants en termes d'efficacité et de temps d'inférence. Le deuxième comparateur de visages que nous avons trouvé durant nos recherches est DeepFace [7]. Cette librairie est basée sur le modèle Visual Geometry Group Face (VGG-Face), un modèle de description de visages développé par VGG et l'université d'Oxford. Les créateurs de DeepFace ont modifié l'architecture du modèle VGG-Face en supprimant la couche de softmax afin de l'utiliser en tant qu'extracteur de caractéristique des visages. Ce modèle est implémenté dans DeepFace grâce à la célèbre librairie Keras.

Pour comparer les visages, cette librairie utilise VGG-Face pour extraire les caractéristiques de chaque visage, puis compare les deux vecteurs obtenus en utilisant par défaut la méthode de calcul de distance cosinus. Le seuil utilisé pour déterminer si les visages appartiennent à la même personne a été fixé à 0.40 par DeepFace.

Après avoir réalisé quelques tests permettant d'évaluer la qualité de cette librairie, il est apparu que son efficacité était très satisfaisante. Son temps d'inférence était quant à lui bien meilleur que celui de Face-Recognition mais toujours trop lent pour notre besoin (environ 150 millisecondes sur une NVIDIA RTX 2060) étant donné que le programme devra comparer plusieurs visages à chaque image.

D. Yolov8-cls

En approfondissant nos recherches, nous avons découvert YOLOv8-cls, une variante du détecteur d'objets YOLOv8 conçu pour de la classification d'images. Ce classifieur pourrait nous être utile afin de différencier les visages d'individus. Le seul problème étant que le modèle doit être entrainé sur un dataset contenant le nom et les visages des personnes pouvant être classifiées. Le projet ne permet pas d'avoir un tel dataset car les vidéos des utilisateurs peuvent contenir les visages de n'importe quelle personne.

Après avoir eu une discussion avec le chercheur et professeur Olivier BROUSSE ainsi que le fondateur d'ultralytics et créateur de YOLOv8 Glenn JOCHER [8] sur les méthodes s'offrant à nous pour la comparaison de visages en utilisant YOLOv8, l'idée nous est venue de

modifier l'architecture de YOLOv8-cls afin de l'utiliser en tant qu'extracteur de caractéristiques.

En effet, le classifieur YOLOv8 est composé d'une couche de softmax comme le montre la figure 5.

```python
class Classify(nn.Module):
    """YOLOv8 classification head, i.e. x(b,c1,20,20) to x(b,c2)."""

    def __init__(self, c1, c2, k=1, s=1, p=None, g=1):
        """Initializes YOLOv8 classification head with specified input and output channels, kernel size, stride,
        padding, and groups.
        """
        super().__init__()
        c_ = 1280  # efficientnet_b0 size
        self.conv = Conv(c1, c_, k, s, p, g)
        self.pool = nn.AdaptiveAvgPool2d(1)  # to x(b,c_,1,1)
        self.drop = nn.Dropout(p=0.0, inplace=True)
        self.linear = nn.Linear(c_, c2)  # to x(b,c2)

    def forward(self, x):
        """Performs a forward pass of the YOLO model on input image data."""
        if isinstance(x, list):
            x = torch.cat(x, 1)
        x = self.conv(x)
        x = self.pool(x)
        x = x.flatten(1)
        x = self.drop(x)
        x = self.linear(x)
        x = x.softmax(1)
        return x
```

Figure 5 - YOLOV8-cls Head

Cette couche permet au modèle de normaliser la sortie de la couche *linear* en une distribution de probabilités représentant la probabilité qu'a l'image d'entrée d'appartenir à chaque classe. En supprimant la couche linéaire dite *fully connected* qui est chargée de mapper les caractéristiques dans l'espace des classes et la couche de softmax, le modèle devient alors un extracteur de caractéristiques idéal pour effectuer du clustering. Le vecteur de caractéristiques en sortie est alors de taille 1280.

Nous avons donc entrainé le modèle YOLOv8-cls sur 1000 individus du dataset VGG-Face contenant chacun entre 300 et 500 visages. Cet entrainement a permis au modèle de bien cerner les caractéristiques dissociant les visages.

Une fois les vecteurs de caractéristiques de deux visages récupérés, il était nécessaire de choisir préalablement un seuil qui permettra une fois la distance cosinus calculée entre les deux images, de savoir si elles correspondent aux mêmes visages ou non.

La distance cosinus entre deux vecteurs A et B est donnée par l'équation (1) :

$$Cosinus\ Distance\ (A,B) = \frac{A.B}{\|A\|.\|B\|} \quad (1)$$

Choisir ce seuil est d'une importance capitale afin de déterminer l'appartenance de visages à une personne. Pour cela, nous avons choisi d'utiliser le même protocole que les créateurs de DeepFace, documenté dans l'article [5]. Cet article explique comment calculer le seuil en réalisant de nombreuses comparaisons comprenant des vrais positifs, des faux positifs, des vrais négatifs ainsi que des faux négatifs.

Après avoir suivi le tutoriel, nous nous sommes malheureusement rendu compte que les 1280 caractéristiques extraites par le modèle modifié YOLOv8-cls n'étais pas assez précise pour permettre de définir si un visage appartenait à une personne ou pas. En effet les courbes des distances des vrais positifs et des faux positifs étaient beaucoup trop proches pour définir un seuil séparant les deux.

E. VGG-Face

Comme expliqué précédemment, la comparaison de visages à l'aide de la librairie DeepFace donnait des résultats très satisfaisant mais avec un temps d'inférence trop long pour notre besoin. Nous avons donc décidé d'explorer en profondeur le code utilisé dans DeepFace afin de voir s'il y avait possibilité d'optimiser certaine partie.

En explorant les classes utilisées par DeepFace, il est apparu que cette librairie utilisait Keras pour implémenter le modèle VGG-Face. Keras est une librairie de haut niveau permettant d'implémenter des réseaux de Deep Learning très rapidement. Cette librairie fournit une couche d'abstraction la rendant accessible aux débutants. En revanche, cette librairie rend les calculs sur le Graphic Processing Unit (GPU) ou carte graphique, particulièrement long du fait de sa couche d'abstraction.

PyTorch est connu pour son graphique de calcul dynamique et son efficace gestion de la mémoire. C'est pour cela que nous avons décidé de réimplémenter ce modèle en PyTorch, afin d'observer la différence des temps d'inférences. Plutôt que de reprogrammer les couches une par une du modèle VGG-Face à l'aide de la librairie PyTorch et de réentraîner le modèle sur le dataset VGG-Face contenant 10000 individus avec un total de 2.6 millions d'images, nous avons décidé d'extraire le modèle Keras de DeepFace. Les deux dernières couches du modèle

VGG-Face sont déjà supprimées dans la librairie DeepFace, permettant son utilisation en tant qu'extracteur de caractéristique. De plus, DeepFace contient les poids du réseau entrainé sur le dataset VGG-Face, nous évitant ainsi de le réentraîner ce qui aurait pris plusieurs jours.

Pour convertir le modèle au format utilisé par PyTorch, nous comme passé par le format Open Neural Network Exchange (ONNX). Ce format permet de représenter des modèles d'apprentissage automatique et a été développer afin de permettre une interopérabilité entre les différentes bibliothèques utilisant chacune leur format de données. Ce format a donc été utilisé afin de passer d'un modèle Keras à un modèle ONNX, puis d'un modèle ONNX à un modèle PyTorch. La figure 6 montre le script utilisé pour passer du format Keras au format PyTorch.

```python
from deepface.basemodels.VGGFace import loadModel
import tensorflow as tf
from onnx import load
from onnx2torch import convert
import torch
import os

### Save Keras model (VGG-Face architecture and weights)
model = loadModel()
model.save('../VGGFace/vgg-face.h5')
model = tf.keras.models.load_model("../VGGFace/vgg-face.h5")
tf.saved_model.save(model, "../VGGFace/tmp_model")

### Keras to ONNX
os.system("python -m tf2onnx.convert --saved-model ../VGGFace/tmp_model --output ../VGGFace/vgg-face.onnx")

### ONNX to PyTorch
onnx_model_path = '../VGGFace/vgg-face.onnx'
onnx_model = load(onnx_model_path)
torch_model = convert(onnx_model)
torch_model_path = '../VGGFace/vgg-face.pt'
torch.save(torch_model, torch_model_path)
```

Figure 6 - Conversion du modèle Keras à PyTorch

Une fois le modèle converti en PyTorch ainsi que ces poids, le temps d'inférence de VGG-Face dans ce nouveau format a été mesuré et il fut assez surprenant d'observer que le passage dans du format Keras au format PyTorch a réduit le temps d'inférence de 80% en passant de 151 ms à seulement 34 ms. La figure 7 illustre le temps d'inférence de VGG-Face sur 160 images aux format Keras, ONNX et PyTorch. Il apparait clairement que la librairie PyTorch fournit d'excellent résultats comparés aux deux autres.

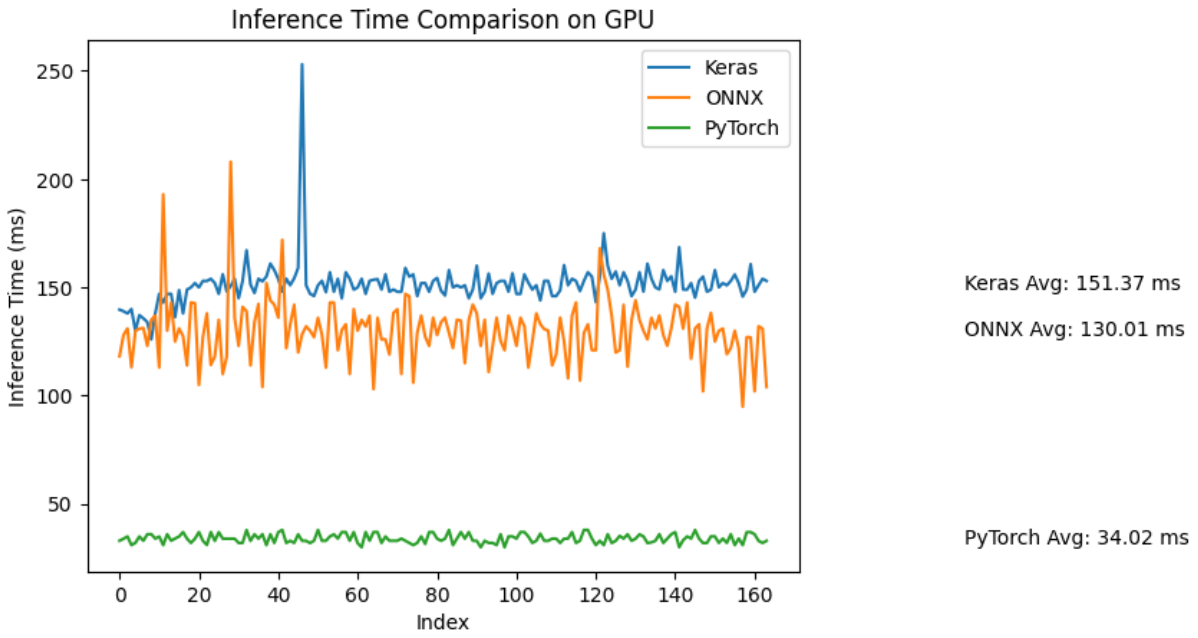

Figure 7 - Benchmark du temps d'inférence de VGG-Face dans différents formats

Le temps d'inférence du modèle était donc satisfaisant dans ce nouveau format, mais il était nécessaire de s'assurer que ses performances en termes d'efficacité à dissocier les individus étaient toujours satisfaisantes et de recalculer le seuil. Pour cela, nous avons suivi l'article [5] expliquant comment calculer le seuil en réalisant un certain nombre de comparaisons de visages. Les résultats de ce calcul de seuil sont disponibles dans le fichier *tests/classifier_tests.ipyb*.

Nous avons tout d'abord sélectionné 17 individus contenant chacun plusieurs visages. Un panda Dataframe a ensuite été créé afin de regrouper les résultats des comparaisons, contenant tout d'abord les colonnes *file_x*, *file_y* et *decision*. Ces 3 colonnes indiquent respectivement le premier visage à comparer, le deuxième visage à comparer et le résultat attendu. Des vrais positifs ont donc été générés en associant deux visages de la même personne, et des vrais négatifs ont été générés en associant deux visages de personnes différentes comme le montre la figure 8.

	file_x	file_y	decision
0	Angelina Jolie\018_fcafe1a8.jpg	Angelina Jolie\022_b497b92e.jpg	Yes
1	Angelina Jolie\018_fcafe1a8.jpg	Angelina Jolie\037_62d00a09.jpg	Yes
2	Angelina Jolie\018_fcafe1a8.jpg	Angelina Jolie\045_c560251e.jpg	Yes
3	Angelina Jolie\018_fcafe1a8.jpg	Angelina Jolie\050_7c5b026c.jpg	Yes
4	Angelina Jolie\018_fcafe1a8.jpg	Angelina Jolie\084_da751ddd.jpg	Yes
...	...	...	...
1979	Tom Hanks\097_0c9b7ced.jpg	Will Smith\046_6d5b1ed6.jpg	No
1980	Tom Hanks\097_0c9b7ced.jpg	Will Smith\051_1f3aede9.jpg	No
1981	Tom Hanks\097_0c9b7ced.jpg	Will Smith\053_8405b30f.jpg	No
1982	Tom Hanks\097_0c9b7ced.jpg	Will Smith\072_4a17b7fb.jpg	No
1983	Tom Hanks\097_0c9b7ced.jpg	Will Smith\087_6eba84a6.jpg	No

Figure 8 - Génération de vrais positifs

Deux colonnes ont ensuite été ajoutées permettant d'enregistrer pour chaque ligne la distance de comparaison des deux visages avec respectivement le modèle Keras de VGG-Face utilisé par DeepFace, et le modèle PyTorch que nous avons généré. Le résultat de ces comparaisons est illustré par la figure 9.

	file_x	file_y	decision	deepface_distance	vgg_face_distance
0	Angelina Jolie\018_fcafe1a8.jpg	Angelina Jolie\022_b497b92e.jpg	Yes	0.366601	0.377351
1	Angelina Jolie\018_fcafe1a8.jpg	Angelina Jolie\037_62d00a09.jpg	Yes	0.423913	0.502540
2	Angelina Jolie\018_fcafe1a8.jpg	Angelina Jolie\045_c560251e.jpg	Yes	0.377955	0.389708
3	Angelina Jolie\018_fcafe1a8.jpg	Angelina Jolie\050_7c5b026c.jpg	Yes	0.362633	0.356034
4	Angelina Jolie\018_fcafe1a8.jpg	Angelina Jolie\084_da751ddd.jpg	Yes	0.377908	0.409797
...	...	...	...	...	...
1979	Tom Hanks\097_0c9b7ced.jpg	Will Smith\046_6d5b1ed6.jpg	No	0.629890	0.692644
1980	Tom Hanks\097_0c9b7ced.jpg	Will Smith\051_1f3aede9.jpg	No	0.652923	0.722766
1981	Tom Hanks\097_0c9b7ced.jpg	Will Smith\053_8405b30f.jpg	No	0.673060	0.768213
1982	Tom Hanks\097_0c9b7ced.jpg	Will Smith\072_4a17b7fb.jpg	No	0.565837	0.644251
1983	Tom Hanks\097_0c9b7ced.jpg	Will Smith\087_6eba84a6.jpg	No	0.611475	0.663956

Figure 9 - Résultats des comparaisons

Ces comparaisons nous ont permis d'obtenir les deux graphiques suivants présentés par la figure 10, illustrant la distribution des distances pour chaque modèle. La courbe bleue représente les distances de comparaisons de visages appartenant à la même personne tandis que la courbe orange représente les distances de comparaisons de visages appartenant à des personnes différentes.

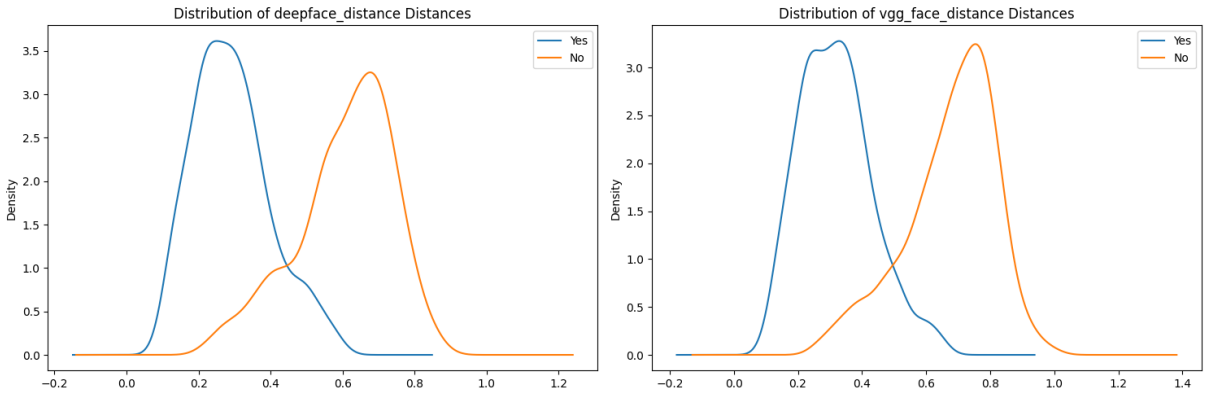

Figure 10 - Distributions des distances des deux modèles

Avec le modèle Keras, un seuil semble se distinguer autour de 0.4 tandis que le modèle PyTorch indique une séparation des deux courbes autour de 5.0. Néanmoins, ces deux courbes montrent que ce modèle n'est pas parfait car certaines distances sont très faibles alors qu'elles sont le résultat de comparaisons de visages n'appartenant pas à la même personne.

Le seuil peut être calculé de 3 manières différentes :

- Un arbre de décision : permet de trouver un seuil optimal qui sépare le mieux les comparaisons de mêmes personnes et de personnes différentes.
- La règle 2-sigma : signifie que les valeurs se trouvent à moins de deux écarts-types de la moyenne dans une distribution. Elle permettrait d'obtenir un seuil sous lequel se trouve 95.45% des distances des comparaisons de visages de la même personne.
- La règle 3-sigma : signifie que les valeurs se trouvent à moins de 3 écarts-types de la moyenne dans une distribution. Elle permettrait d'obtenir un seuil sous lequel se trouve 99.73% des distances des comparaisons de visages de la même personne.

Afin d'identifier le meilleur seuil, nous les avons calculés tous les 3 pour les deux modèles et nous avons obtenu les matrices de confusion représentées par la figure 11.

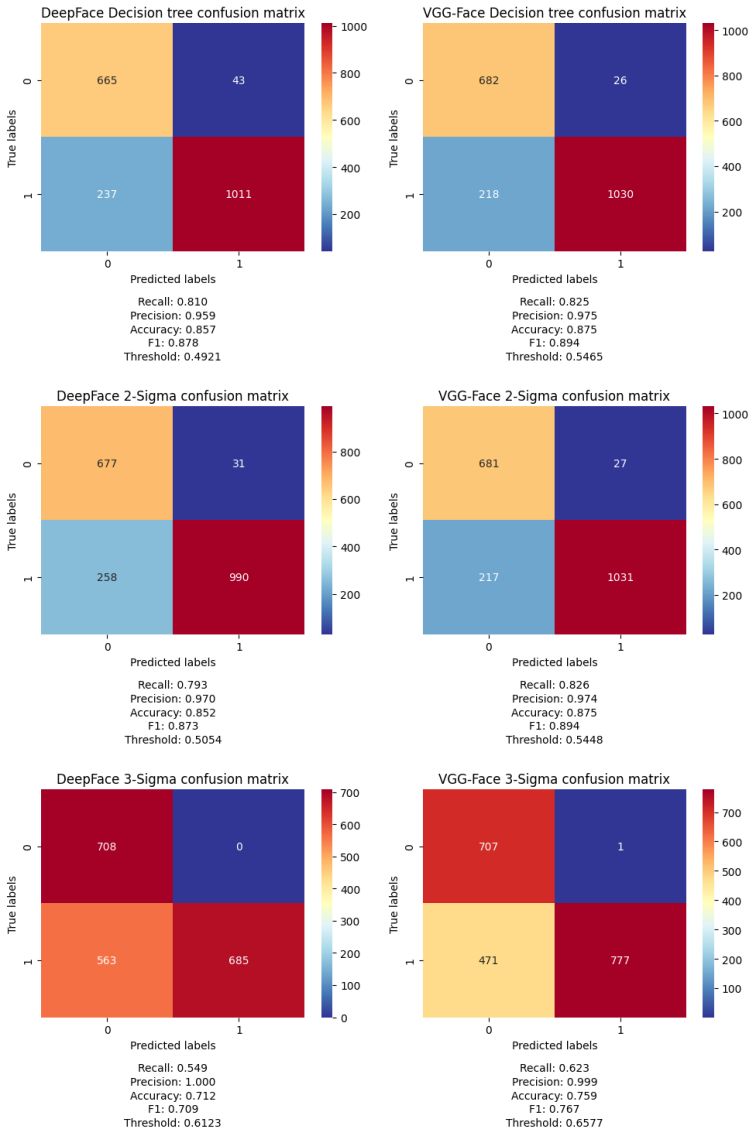

Figure 11 - Matrices de confusions des 3 seuils pour les 2 modèles

Pour le modèle PyTorch, les seuils calculés par les méthodes arbre de décision, 2-sigma et 3-sigma sont respectivement 0.55, 0.54 et 0.65. Comme le montre la figure 10, plus le seuil est élevé, plus le nombre de faux positifs (visages considérés par erreur comme appartenant à la même personne) est important. En revanche, plus le seuil est faible, plus le nombre de faux négatifs important.

Étant donné la confusion des deux courbes dans la distribution des distances, il était clair qu'il n'était pas possible d'obtenir un seuil parfait. C'est pour cette raison que nous avons décidé de développer un algorithme de correction, qui nous permettra de diminuer le nombre de faux négatifs. Nous avons donc choisi d'utiliser un seuil élevé au sein du modèle qui génèrera

dans le pire des cas des faux positifs qui seront corrigés par la suite. L'algorithme de comparaison est détaillé dans la partie 5 de ce chapitre.

3. Implémentation du FaceComparator

L'implémentation du comparateur de visages repose sur la classe *VGGFaceComparator*, qui hérite à la fois de la classe abstraite *Model* et de l'interface *FaceComparator*. Cette classe utilise le modèle VGG-Face PyTorch pré-entraîné pour extraire les caractéristiques des visages et effectuer une comparaison basée sur la distance cosinus.

Dans le constructeur de la classe, le chemin du modèle VGG-Face PyTorch est fourni, le modèle est chargé, et le dispositif d'exécution (CPU ou GPU) est déterminé. Le modèle est ensuite déplacé sur le dispositif approprié (GPU s'il est disponible). Le seuil de similarité (*threshold*) est défini à 0.40, et la méthode *warm_up* est appelée.

La méthode *warm_up* prend une image vide, la pré-traite à l'aide de la méthode *image_preprocess*, et effectue une prédiction avec le modèle VGG-Face. Cela assure que le modèle est initialisé et prêt pour les comparaisons subséquentes.

La méthode *image_preprocess* effectue le prétraitement nécessaire sur une image donnée avant de l'entrer dans le modèle VGG-Face. Cela inclut le redimensionnement de l'image, la conversion en format float32, et l'ajout d'une dimension supplémentaire.

La méthode *compare* est utilisée pour comparer deux visages. Les visages d'entrée sont pré-traités par la méthode *image_preprocess*, les caractéristiques des visages sont extraites à l'aide du modèle VGG-Face, et la distance cosinus entre ces caractéristiques est calculée. En fonction de cette distance, la méthode *_is_same_person* détermine si les visages appartiennent, ou non, à la même personne.

Les méthodes *_findCosineDistance* et *_is_same_person* sont des méthodes auxiliaires pour calculer la distance cosinus entre les caractéristiques des visages et déterminer si les visages appartiennent à la même personne en fonction du seuil défini, respectivement.

Cette implémentation fournit une approche robuste pour la comparaison de visages en utilisant le modèle VGG-Face et offre une structure modulaire qui peut être étendue à de nouvelles fonctionnalités mais également à d'autres comparateurs de visages en fonction des besoins du projet.

4. Analyse d'une frame de vidéo

L'analyse d'une frame de vidéo se déroule dans la classe *PersonManager*. La méthode *analyze_frame* est appelée pour chaque frame, ce qui déclenche la détection des visages à l'aide du détecteur de visages (*FaceDetector*). Les visages détectés sont ensuite analysés et associés aux personnes existantes ou ajoutés en tant que nouvelles personnes.

Analyse d'une Frame de Vidéo :

La méthode *analyze_frame* prend en paramètre l'indice de la frame dans la vidéo ainsi que la frame elle-même. Elle commence par détecter les visages dans la frame à l'aide du détecteur de visages (*face_detector.detect*). Les prédictions obtenues sont utilisées pour créer des instances de la classe *Face* représentant chaque visage détecté, en stockant l'index de la frame, la bounding box du visage détecté et son score de confiance (objet *Prediction*).

Pour chaque visage détecté, la méthode *_save_face* est appelée pour attribuer le visage à une personne. Les étapes suivantes sont effectuées pour chaque visage détecté :

- Sauvegarde du Visage : La méthode *_save_face* prend le visage détecté, la frame et une liste d'identifiants de personnes dans la frame actuelle. Elle recadre le visage à l'aide de l'éditeur d'image (*ImageEditor.crop*) et obtient les caractéristiques du visage recadré à l'aide du comparateur de visages (*face_comparator.get_features*).
- Vérification de l'Existence de Personnes : Si aucune personne n'a été identifiée jusqu'à présent (*_is_persons_empty*), une nouvelle personne est créée et le visage est associé à cette personne en tant que visage de référence (attribut *cropped_face* de la classe *Person*).
- Comparaison avec les Personnes Existantes : Si des personnes existent déjà, la méthode parcourt chaque personne pour comparer le visage avec les visages existants. La comparaison est basée sur les caractéristiques du visage recadré (*cropped_face_features*) et les caractéristiques du visage de référence de la personne. Si une personne existante correspond au visage actuel, le visage est ajouté à cette personne. Sinon, une nouvelle personne est créée. Dans le cas où le visage a été associé à une personne déjà existante, le visage de référence de cette personne peut être remplacé par le visage actuel si le score de confiance de ce dernier est plus élevé.
- Retour de l'Identifiant de la Personne : L'identifiant de la personne (UUID) associée au visage est renvoyé pour éviter d'ajouter plusieurs visages d'une frame à une même

personne. En effet, afin d'améliorer les performances de l'algorithme, nous avons émis l'hypothèse qu'une personne ne peut être présent deux fois sur la même frame.

Comparaison de Visages et de Features :

La classe *PersonManager* expose également des méthodes telles que *compare_faces* et *compare_features* pour la comparaison de visages et de caractéristiques respectivement. Ces méthodes utilisent le comparateur de visages (*face_comparator*) pour effectuer les comparaisons nécessaires.

En résumé, l'analyse d'une frame de vidéo s'appuie sur la détection de visages, la comparaison avec les visages existants, et l'association des visages aux personnes. Les comparaisons sont effectuées en utilisant les caractéristiques extraites des visages à l'aide du comparateur de visages. Enfin, les personnes sont regroupées en fonction de ces comparaisons pour améliorer la cohérence de l'identification des personnes.

5. Algorithme de correction

Afin d'améliorer la précision de nos résultats pour les médias vidéo, un algorithme de correction a été développé. Cet algorithme, une fois la vidéo entièrement traitée (détection et classification des visages), analyse les résultats des comparaisons et applique des corrections pour minimiser les faux positifs.

L'algorithme se déroule dans deux parties principales : une dans person_manager.py et l'autre dans video_processor.py.

person_manager.py :

Le *PersonManager* expose la méthode *group_identical_persons* qui itère sur chaque paire de personnes et utilise le comparateur de visages pour déterminer si deux personnes sont identiques. Si tel est le cas, elles sont fusionnées en appelant la méthode *_merge_persons*.

La méthode *_merge_persons* compare les scores de confiances des visages de référence *cropped_face* des deux personnes. La personne avec un meilleur score de confiance conserve sa liste d'objets *Face* (pour rappel *Face* possède un index indiquant l'image à laquelle a été détecté le visage ainsi qu'un objet Prédiction) et ajoute la liste des objets *Face* de l'autre personne à la sienne. Ensuite, la personne avec une moindre confiance est retirée de la liste des personnes, garantissant que chaque personne conserve le visage le plus fiable.

video_processor.py :

Le processus de correction commence en appelant la méthode *_correction* du *VideoProcessor*. À ce stade, le gestionnaire de personnes (*PersonManager*) a déjà entièrement traité la vidéo (détection et classification des visages) à l'aide du détecteur et du comparateur de visages.

Tout d'abord, le regroupement des personnes identiques est effectué en appelant la méthode *group_identical_persons* de *PersonManager*.

Ensuite, dans la boucle principale, chaque visage détecté pour chaque personne est pris en compte. Pour chaque visage, une comparaison est effectuée avec tous les visages des autres personnes. Si la comparaison indique que le visage actuel et le visage de l'autre personne sont identiques (*comparison.is_same_person* est vrai), une vérification supplémentaire est effectuée.

Si le visage de l'autre personne a une distance plus faible par rapport au visage actuel que le visage actuel par rapport au visage de référence de la personne d'origine, cela signifie que le visage de l'autre personne a une meilleure correspondance avec le visage actuel que la personne attribuée à l'origine pour ce visage. Dans ce cas, le visage est transféré à l'autre personne, et le visage actuel est retiré de sa personne d'origine.

Cet algorithme de correction contribue à améliorer la précision de l'identification des personnes en associant correctement les visages détectés aux personnes auxquelles ils appartiennent, tout en tenant compte de la confiance associée à chaque visage.

6. Conclusion

Ce chapitre a exploré en détail la reconnaissance faciale, en mettant l'accent sur les méthodes de comparaison de visages. Nous avons présenté des bibliothèques populaires telles que face_recognition, DeepFace ou encore Yolov8-cls. Finalement, nous avons intégré dans notre FaceComparator un modèle VGG-Face, provenant de DeepFace et convertit en PyTorch afin d'optimiser les performances de notre application.

De plus, l'algorithme de correction a été introduit comme un élément clé pour renforcer la fiabilité des résultats. Cette approche complète, combinant un modèle de reconnaissance faciale de pointe et une optimisation personnalisée, constitue la base de notre capacité à regrouper efficacement les visages détectés en individus distincts.

VII. Chapitre 4 : Interface utilisateur

1. Introduction

L'interface utilisateur occupe une place importante dans notre projet, permettant de faire le lien entre la complexité algorithmique et l'utilisateur final. Dans ce chapitre, nous explorerons l'aspect de notre solution grâce à l'intégration de Gradio, une bibliothèque Python puissante facilitant la création d'interfaces interactives pour les modèles d'intelligence artificielle.

Grâce à son approche simplifiée, Gradio offre une expérience utilisateur intuitive. Notre choix de cette bibliothèque découle de sa capacité à rendre accessible la puissance de l'intelligence artificielle sans compromettre la sophistication de notre solution.

Nous mettrons en lumière les différentes fonctionnalités de l'interface, du téléchargement de vidéos/images à la sélection des individus à flouter, tout en mettant en avant les aspects ergonomiques et les choix de conception qui optimisent l'expérience utilisateur.

2. Présentation de l'interface

Notre interface utilisateur, conçue pour être simple et efficace, se divise en deux onglets distincts, chacun dédié à un type de fichiers spécifique : *Video Processing* pour le traitement des vidéos et *Image Processing* pour le traitement des images.

Onglet Video Processing :

Lorsque les utilisateurs accèdent à cet onglet, ils sont accueillis par une interface épurée qui permet de télécharger facilement une vidéo depuis leur appareil. L'utilisateur peut visualiser un aperçu de la vidéo qu'il vient de déposer.

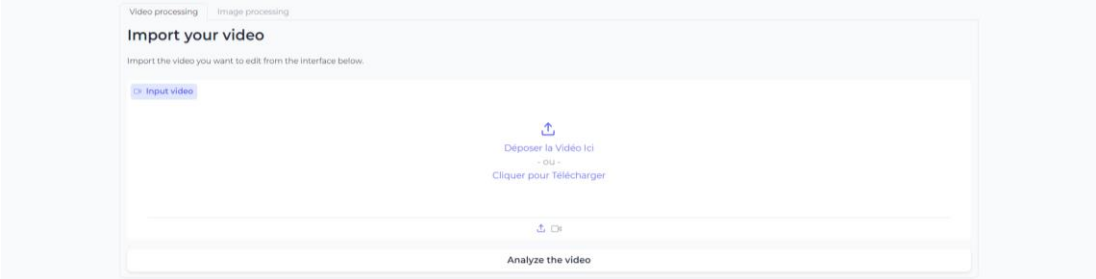

Figure 12 - Onglet "Video processing" de l'interface (1)

Sous cet aperçu, un bouton "*Analyze the video*" permet à l'utilisateur de lancer le processus de détection des visages, suivi d'une barre de progression pour indiquer l'état d'avancement. Le processus détecte les visages présents dans la vidéo, les regroupe selon les individus, et offre une liste intuitive de ces individus à l'utilisateur. Pour chaque individu, une checkbox permet ou non de rendre anonyme le visage de cette personne dans toute la vidéo. Une checkbox permet aussi à l'utilisateur de choisir entre un flou « dégradé » et un flou brut.

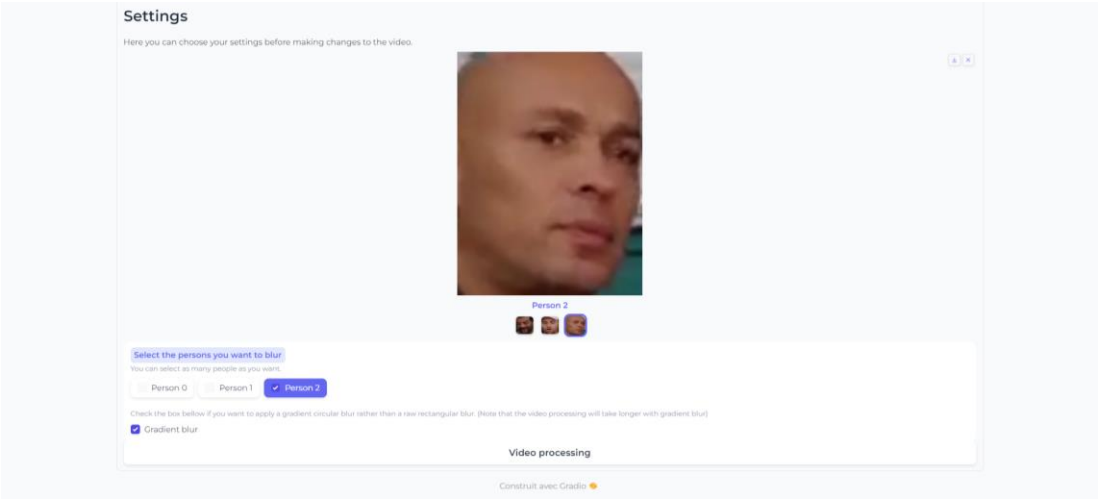

Figure 13 - Onglet "Video processing" de l'interface (2)

Une fois les choix de l'utilisateur effectués, un bouton "*Video processing*" permet de lancer le processus de floutage des visages sélectionnés, suivi d'une barre de progression pour indiquer l'état d'avancement. Quand le processus de floutage est terminé, l'utilisateur peut visualiser et télécharger le résultat vidéo.

Figure 14 - Onglet "Video processing" de l'interface (3)

Onglet Image Processing :

Cet onglet est dédié au traitement des images. Il propose une interface similaire à celle de Video Processing mais optimisée pour les images statiques. Les utilisateurs peuvent télécharger une image et, après traitement, visualiser le résultat avec la possibilité de flouter les visages détectés identiquement à l'interface des vidéos.

L'aspect convivial de l'interface se manifeste par la simplicité de ces deux onglets. L'utilisateur n'est pas submergé par des options complexes, mais plutôt guidé de manière intuitive à travers le processus de traitement, offrant ainsi une expérience utilisateur fluide et accessible.

3. Implémentation

L'implémentation de notre interface utilisateur repose sur l'utilisation de la bibliothèque Gradio, choisie pour sa facilité d'intégration à notre projet.

Voici comment nous avons mis en œuvre chaque onglet de notre interface :

Fichier main.py :

Le fichier principal main.py crée une interface tabulée utilisant Gradio. Deux onglets sont inclus, un pour le traitement vidéo (Video processing) et un autre pour le traitement d'image (Image processing). Le code utilise également un thème « doux » (gr.themes.Soft()).

Fichier video_interface.py :

Initialisation de l'interface de Vidéo (VideoInterface) :

- *__init__ :* Initialise la classe avec un objet Blocks de Gradio, un processeur vidéo, et une liste vide de personsDTO.

Définition des onglets (part1, part2, part3) :

- *part1_video_container :* Section pour importer la vidéo.
- *part2_video_container :* Section pour les paramètres de traitement vidéo.
- *part3_video_container :* Section pour la sortie vidéo.

Gestion des événements :

- *analyse_video :* Méthode pour analyser la vidéo et obtenir les visages détectés.

- *update_persons_should_be_blurred :* Méthode pour mettre à jour les personnes devant être floues.
- *apply_blur :* Méthode pour appliquer le flou à la vidéo.
- *reset :* Réinitialise le processeur vidéo et la liste personsDTO.

Fichier image_interface.py :

Initialisation de l'interface d'Image (ImageInterface) :

- *__init__ :* Initialise la classe avec un objet Blocks de Gradio, un processeur d'image, et une liste vide de personsDTO.

Définition des onglets (part1, part2, part3) :

- *part1_image_container :* Section pour importer l'image.
- *part2_image_container :* Section pour les paramètres de traitement d'image.
- *part3_image_container :* Section pour la sortie d'image.

Gestion des événements :

- *analyse_image :* Méthode pour analyser l'image et obtenir les visages détectés.
- *update_persons_should_be_blurred :* Méthode pour mettre à jour les personnes devant être floues.
- *apply_blur :* Méthode pour appliquer le flou à l'image.
- *reset :* Réinitialise le processeur d'image et la liste personsDTO.

Remarques générales :

La gestion des sections (part2 et part3 pour chaque interface) est basée sur la modification des styles CSS pour afficher ou masquer ces parties en fonction des actions de l'utilisateur.

Les fonctions *analyse_video* et *analyse_image* utilisent les processeurs vidéo et image respectifs (*VideoProcessor* et *ImageProcessor*) pour obtenir des informations sur les personnes détectées dans la vidéo ou l'image.

La fonction *update_persons_should_be_blurred* modifie les *personsDTO* en fonction des sélections de l'utilisateur et *apply_blur* appelle la fonction de réalisation du flou de *VideoProcessor* ou *ImageProcessor* en lui passant en paramètre les *personsDTO*, entre autres.

4. Conclusion

En concluant notre exploration de l'interface utilisateur, il est évident que l'intégration de Gradio a joué un rôle prépondérant dans son succès.

Grâce à cette plateforme, nous avons pu concevoir une interface conviviale, offrant aux utilisateurs une expérience simplifiée tout en assurant une gestion efficace du processus de floutage des visages. La facilité d'interaction avec le programme, permettant de télécharger, analyser et anonymiser les visages en quelques clics, met en évidence l'efficacité de notre solution pour atteindre nos objectifs initiaux.

En résumé, ce chapitre souligne la symbiose entre la convivialité de notre interface utilisateur et l'implémentation de notre solution d'intelligence artificielle.

VIII. Conclusion Générale

En résumé, notre projet a abouti à la création d'une interface conviviale permettant le floutage sélectif de personnes dans des images et des vidéos. L'utilisation de Gradio a été cruciale pour développer une interface interactive et intuitive, offrant deux onglets distincts pour le traitement d'images et de vidéos.

La modularité de notre approche a simplifié la maintenance et l'extension du code. La détection précise des visages et la possibilité de flouter sélectivement des personnes sont des points forts de notre solution, offrant aux utilisateurs une flexibilité appréciable.

Bien que notre projet réponde aux objectifs fixés, des améliorations futures pourraient être envisagées comme perfectionner le comparateur de visages et accélérer la réalisation du flou « dégradé » ainsi que le processus de correction des visages détectés pour les vidéos. Cependant, dans l'ensemble, notre application représente une exploration réussie de la vision par ordinateur et de l'interaction utilisateur.

En conclusion, notre projet offre une solution fonctionnelle et attrayante facilitant le traitement interactif des médias visuels. Il répond de manière directe à notre problématique de base : « Comment le développement de telles technologies (enregistrements vidéo de plus en plus constants) peut coexister avec les normes éthiques et légales ? ». Notre application offre alors une approche pratique en réduisant à seulement quelques clics la gestion délicate du droit à l'image, en particulier lorsqu'il s'agit de personnes présentes à leur insu dans des vidéos ou des images.

IX. Sources

[1]. 17/09/2017 Robert C. MARTIN– « Clean Architecture ».

[2]. Gaudenz Boesch – « Object Detection in 2024 : The Definitive Guide », rédigé le 12/04/2023.
Article disponible à l'adresse : https://viso.ai/deep-learning/object-detection/

[3]. Ultralytics – « Ultralytics YOLOv8 », dernière modification le 28/12/2023.
Article disponible à l'adresse : https://github.com/ultralytics/ultralytics

[4]. Fares ELMENSHAWII – « Face-Detection-Dataset », dernière modification le 01/06/2023.
Dataset disponible à l'adresse :
https://www.kaggle.com/datasets/fareselmenshawii/face-detection-dataset/

[5]. Sefik ILKIN SERENGIL – « Fine Tuning the Threshold in Face Recognition », rédigé le 22/05/2020.
Article disponible à l'adresse : https://sefiks.com/2020/05/22/fine-tuning-the-threshold-in-face-recognition/

[6]. Adam GEITGEY – « face_recongition », dernière modification le 10/06/2022.
Librairie disponible à l'adresse : https://github.com/ageitgey/face_recognition

[7]. Sefik ILKIN SERENGIL – « DeepFace », dernière modification le 02/01/2024.
Librairie disponible à l'adresse : https://github.com/serengil/deepface

[8]. Glenn JOCHER – « Glenn JOCHER », dernière modification le 19/11/2023.
Page GitHub disponible à l'adresse : https://github.com/glenn-jocher