ARTIFICAL INTELLIGENCE

PROJECT REPORT

Authors:

EL ALOUANI Naofel

COUTAREL Allan

ESIREM 5A INFOTRONIC – ILC GROUP TP2

Year 2023/2024

Contents

I. List of Figures

II. Acronyms

CNN	Convolutional Neural Network
CPU	Central processing unit
D.L.	Deep Learning
DPM	Deformable Part Model
DTO	Data Transfer Object
FPS	Frame Per Second
Go	Gigabyte
GPU	Graphics Processing Unit
IDE	Development environment
MS	Milliseconds
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. General Introduction

We live in an era where image processing and facial recognition are at the heart of many fields such as security, health, marketing and entertainment. Video recording is becoming more and more common thanks to new technologies integrated into smartphones.

However, it raises new questions regarding image rights. For example, during reporting or interviews in public places. On the one hand, there is the need to inform an audience which is the very essence of all media. On the other hand there is the need to respect the privacy of individuals present without their knowledge on a video. A problem therefore arises: How can the development of such technologies coexist with ethical and legal standards?

Our project provides a solution to this problem, allowing in 4 clicks to avoid any prosecution linked to respecting a person's image rights when publishing a photo or video. This project report is structured into four distinct chapters to address this question. The first chapter details the specifics of the project, including its architecture, the algorithm developed, as well as the technologies involved. The second chapter focuses on the face detector, exploring the state of the art of object detectors and the model training and inference processes. The third chapter deals with face comparison, examining different techniques and tools commonly used in this field. Finally, the fourth chapter presents the user interface, crucial for effective interaction with the program, before concluding with a general summary which will review the challenges and prospects of this ambitious project.

IV. Chapter 1: Presentation of the project

1. Introduction

Through this chapter, we will explore the design of the project. In order to properly target the functionalities and business needs of the project, we will begin our exploration with a detailed analysis of the strategic choices in terms of technical solutions, highlighting the reasons underlying these decisions and their impact on the main objective. of the project.

Next, we will dive into the study of the project architecture which was carefully developed before programming began. This section will emphasize the importance of an optimized architecture for the realization of a robust project that is insensitive to the addition of new functionalities. We will study the technologies and tools used to successfully complete the development of the project.

2. Features

The problem that gave rise to this project is respect for image rights. Indeed, it is essential to respect the image rights of individuals present during a video recording in a public place so that they are not filmed without their knowledge.

In order to guarantee the anonymity of each individual filmed without their knowledge, different solutions can be put in place. One of the most effective and commonly used is face blurring. This is the technical solution that we have chosen to adopt within our project in order to guarantee the anonymity of each viewer of a video.

However, blurring the same face on each frame of a video is a particularly long and non-trivial process for a user. This is why the second key functionality of our project is individual recognition. Indeed, our project aims to detect each face present on a video and group them according to individuals. This solution will therefore allow a user:

i. To send a video which will be analyzed by the program in order to detect all the faces and group them according to individuals.
ii. To select the individuals to blur
iii. To download the video with the blurred individuals

The facial blurring of each face of the same person will therefore be automatic and entirely carried out by our program. The user will only have to upload their video and choose the people to blur.

3. Project architecture

In order to guarantee seamless robustness and scalability, the project architecture was developed before development even began. This strategic decision is crucial to avoid dispersion of efforts and ensure focus on essential elements. It contributes to meticulous planning and the development of an agile infrastructure, ready to flexibly adjust to future changes and expansions. The efficiency of the architecture is directly reflected in the robustness and adaptability of the development in the face of the challenges encountered. As programming expert Robert C. Martin points out:

> *"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]

It is following this philosophy that we designed the program by adopting the architecture of the Unified Modeling Language (UML) diagram presented in Figure 1.

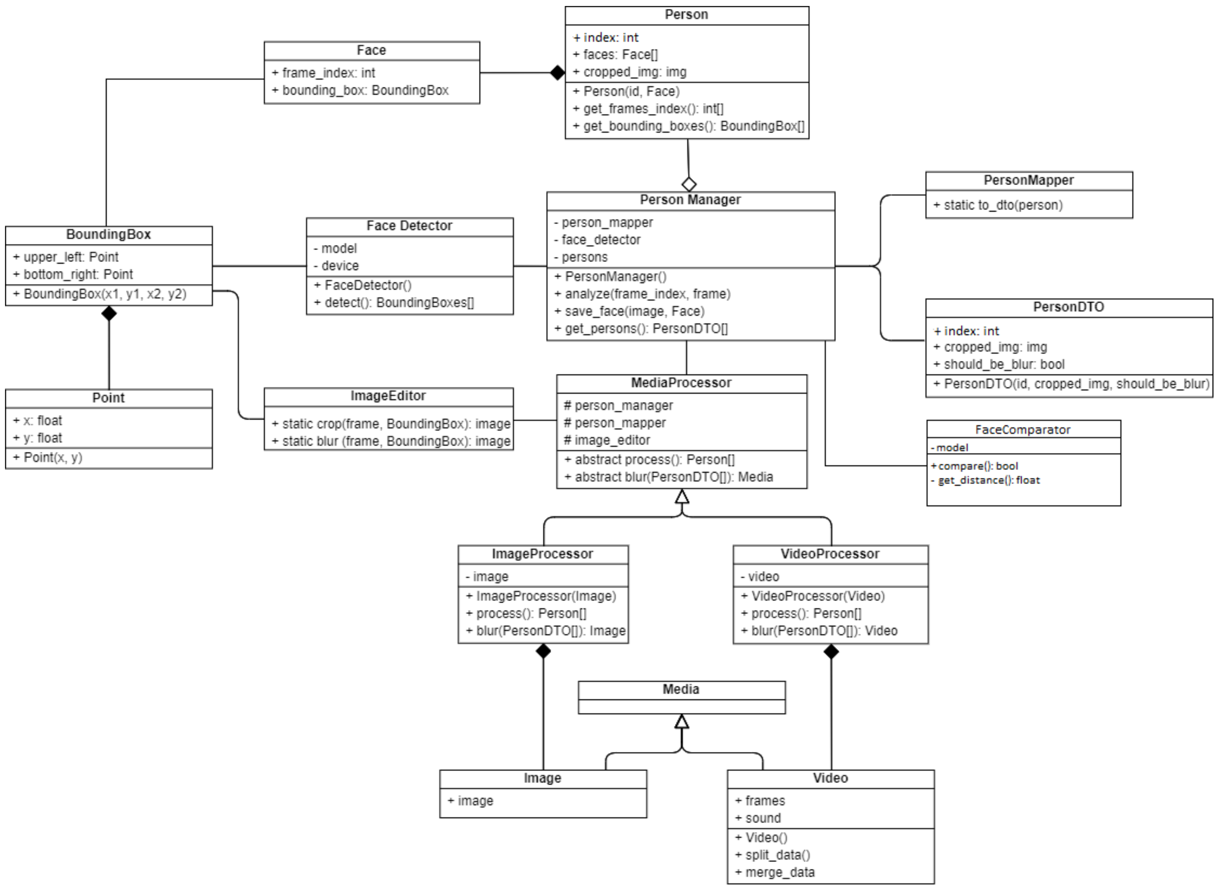


Figure1- UML diagram of the project architecture

This architecture has been carefully designed to facilitate the development of this software while optimizing memory as much as possible. Some classes were slightly modified during the development of the project but will be explained below.

Here is a brief description of the classes shown in Figure 1:

- **BoundingBox**: This class contains two Points. Which correspond to coordinates. A BoundingBox object is created when a face has been detected. The coordinates of this box indicate the coordinates of the face in the image.
- **Comparison**: The Comparison class is not present on the diagram and plays an interface role. A Comparison object is returned after comparing two faces. It contains a Boolean variable is_same_person as well as a distance variable indicating the distance between the two faces.
- **Face**: The Face class has an index indicating the image in which the face was detected as well as a Prediction object which will be detailed later.

- **FaceDetector**: This class is responsible for performing face detection. It includes a detect method, a warm_up method as well as a model attribute representing the neural network responsible for face detection.
- **FaceComparator**: The FaceComparator class is an abstract class responsible for comparing faces to find out if two faces represent the same people. Several classes inherited from this class because we implemented several means of comparison. All this will be detailed in chapter 3.
- **Picture**: The Image class inherits from the Media class. It allows the software to also process images.
- **ImageEditor**: This class is made up of static methods allowing image processing such as cutting or even blurring an area defined by a BoundingBox.
- **Media**: The Media class is abstract and has two child classes: the Image class and the Video class.
- **MediaProcessor**: MediaProcessor is also an abstract class. It is the parent class of ImageProcessor and VideoProcessor.
- **Person**: The Person class represents an individual. An individual contains an id, a list of Faces, a cropped_face which represents the reference face of this person. The cropped_face is used during comparisons to know if a face belongs to a person. It also contains a cropped_face_confidence representing the quality of the cropped_face.
- **PersonDTO**: a Data Transfer Object (DTO) is an object used to simplify the transfer of an object. It is used here to send the individuals detected in the video to the frontend.
- **PersonManager**: The PersonManager class is the key class for grouping faces according to individuals. It is responsible for assigning a face to an individual.
- **PersonMapper**: This class has a static method for converting an object of type Person into PersonDTO.
- **Point**: The Point class represents, as its name suggests, a Point. It is used in the BoundingBox class to define the upper left and lower right corner of the box.
- **Prediction**: The Prediction class is used as output for face detection. It contains a BoundingBox as well as a confidence score used to define the quality of the detected face.
- **Video**: The Video class is a class used to load a video. It contains the path to the video, the number of frames per second, the audio of the video and the total number of frames

in the video. This class also contains methods allowing you to interact with the video such as retrieving the nth image or extracting the audio track.

- **VideoProcessor**: This class is responsible for analyzing the video and correcting and saving the output video.

4. Technologies used (Git for versioning etc.)

For the development of the project, we chose Visual Studio Code (VS Code) as a development environment (IDE). VS Code is a free code editor developed by Microsoft offering a multitude of extensions.

In order to properly manage the dependencies of our project we set up a Python virtual environment using venv. The use of venv was essential to ensure that each team member works in an isolated and controlled environment, which is essential to avoid dependency conflicts with previously completed projects.

Regarding versioning, we have integrated Git into our organization. Git is an essential tool for tracking changes, collaboration and version management in a project. It allowed us to maintain a detailed history of changes, which is vital for understanding the evolution of the project.

Each of these tools was chosen not only for their individual features, but also for how they complement and integrate together.

5. Conclusion

This chapter allowed us to introduce the needs of the project, its key functionalities of the project and to present its architecture. We studied the needs related to respecting image rights and how our face blurring solution, combined with sophisticated individual recognition, meets these needs while making the task easier for the end user. The carefully designed project architecture serves as a solid foundation, ensuring not only robustness but also flexibility to accommodate future expansions and modifications.

Chapter 2 will present the study and development of the face detector which plays a major role in the successful completion of the project.

V. Chapter 2: Face Detector

1. Introduction

The first chapter made it possible to introduce the need which gave rise to this project as well as all the details linked to the organization of its development.

We will begin this new chapter with a state of the art of object detectors in the world of neural networks in order to analyze the solutions available to us for detecting faces in an image. This state of the art will be followed by training of the chosen model then an explanation of its implementation within the project.

2. State of the art of object detectors

A. Definition

It is first necessary to define what object detectors are and why their role is crucial within our project.

Object detection in Deep Learning (DL) is a process aimed at detecting visual classes in an image. Neural networks designed for object detection generally provide the coordinates of a box on the image containing the object. This box is often accompanied by a confidence score indicating the probability that the detected object belongs to the returned class.

The goal of object detection is to have the highest possible accuracy while having the lowest possible inference time. An inference is the term used to describe the realization of a prediction by a neural network.

Object detection is used today in various fields such as security, industry, robotics and many more.

B. Historical

Object detection has evolved enormously over the last 20 years. Early approaches to object detection. Its evolution is divided into three eras[2]:

- Before 2014, the era of traditional object detection
  - Viola-Jones Detector – 2001: This detector is one of the first effective object detectors. This detector was mainly used for face detection and then became a basis for many detection methods.

  - Deformable Part Model (DPM) – 2008: this model introduced bounding box regression which subsequently became a central element of object detection.
- After 2014, The era of object detection after Deep Learning
  - Region based Convolutional Neural Network (R-CNN) – 2014: this algorithm introduced the use of Convolutional Neural Network (CNN) for object detection which made it possible to take a new step in terms of precision.
  - Mask R-CNN – 2017: this model is an evolution of Faster R-CNN (2015) which stands out thanks to its ability to perform not only object detection, but also instance segmentation. Instance segmentation is a concept aimed at cutting each object pixel by pixel unlike object detection which defines a rectangular area in which the detected object is located.
- You Only Looke Once (YOLO) – 2016
  - The YOLO model marked the beginning of a new era: as the name suggests, the model analyzes the image only once, making it very fast and introducing real-time detection.

As we saw in the previous history, there are two categories of object detection methods: two-step and one-step detections. These detectors solve two essential tasks for object detection:

1. Find number of objects
2. Classify each object and estimate its size

Two-stage detectors (R-CNN for example) start calculating the approximate regions of the bounding boxes using the features extracted from the image and then use them to classify the object and perform bounding box regression. Their architecture is therefore composed of two stages: a first which is responsible for approximately defining the region of an object and the second which classifies the object and refines the bounding box. This architecture makes two-stage detectors detectors with high precision but their inference time is generally longer.

One-step detectors such as YOLO, for example, predict bounding boxes without going through the step of defining a region for the object. This process is therefore much faster and allows these models to perform object detection in real time. Their small weakness is that they are not very efficient at recognizing irregularly shaped objects unlike two-step detectors.

C. Today

YOLO models have continued to evolve until today. The latest YOLO model is YOLOv8, developed by the creators of YOLOv5. This model is open-source and can perform four different tasks:

- Object detection
- Instance segmentation
- Image classification
- Pose estimation allows you to determine the position and orientation of the elements of an object (often a human)

YOLOv8 exists in 5 versions: n, s, m, l, x. These versions vary in their number of parameters ranging from a few million parameters for the YOLOv8-n version to 70 million for the YOLOv8-x version. The performance comparison of the YOLOv8 versions is shown in Figure 2.

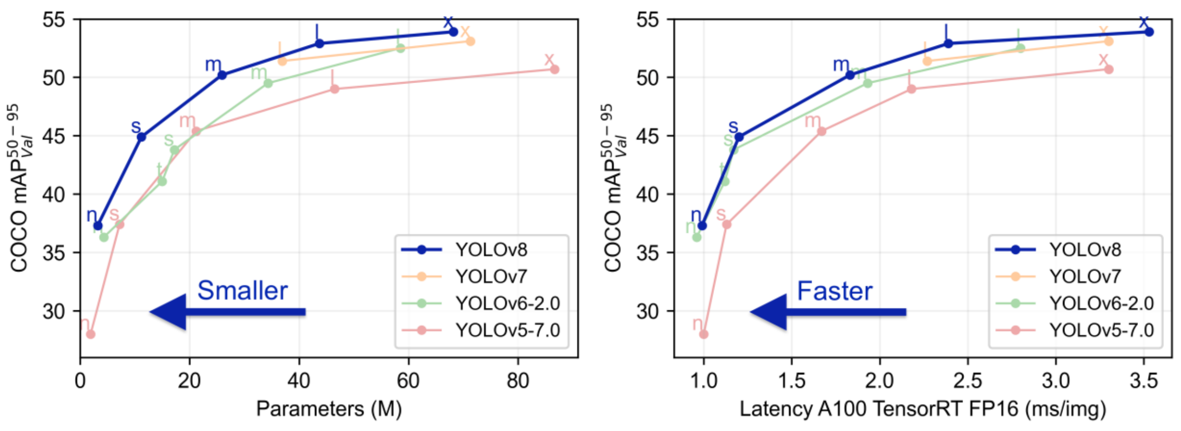


Figure2- Performance of YOLOv8 versions[3]

We notice that the lighter the model, the lower its inference time, reaching one millisecond (ms) for version n. On the other hand, the heavier the model, the more precise it is.

To achieve fast and efficient face detection, essential for ensuring a smooth and responsive user experience, we chose YOLOv8-n. By optimizing the model for speed, we aim to minimize the delay between reading an image and detecting faces.

3. Model training

The developers of YOLOv8 placed it in a library, creating an entire ecosystem around the model in order to have an abstraction layer that allows users to easily take control of this neural network. This library is called ultralytics [3] and available for free via GitHub or the pip utility.

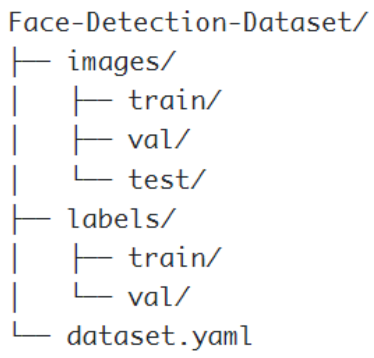
This library provides a function allowing you to train YOLOv8 on a custom dataset.

D. Choice of dataset

In order to have the most precise face detection possible, it is necessary to prepare a complete dataset on which the YOLOv8 model can be trained. After carrying out some research on the Kaggle and Roboflow sites which are universal references in terms of dataset libraries in the world, a dataset seemed ideal for our needs. Indeed, the Face-Detection-Dataset dataset[4]brings together no less than 16.7k images of faces for only 5 Gigabytes (GB). Each image is annotated and associated with a text file containing the coordinates of the bounding box present on the image.

To be able to serve as a training base for a YOLO detector, the dataset requires a specific architecture called YOLO. The root folder of the dataset should contain two main folders, "images" and "labels". The "images" folder contains all the images in the dataset, while the "labels" folder contains the label files associated with the images. Inside the "images" folder, images are typically split into separate subfolders such as "train", "val" and "test", corresponding to training, validation and test data respectively. Each image in the “images” folder has a corresponding label file in the “labels” folder as a .txt file with the same name as the image. These label files contain object annotations in YOLO format, specifying the coordinates of the bounding boxes and the classes of the objects present in the images. In addition to these folders, a dataset.yaml configuration file is necessary in order to describe the details of the dataset, including the paths to the image and label folders , as well as other essential parameters such as the object classes used or the total number of images.

The dataset we used was not in YOLO format, so a small Python script allowed us to put it in the correct architecture.



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

Figure3- Architecture of the dataset in YOLO format

E. Training

The dataset is now ready, so the YOLOv8-n model can be trained on it. To do this, the Ultralytics library provides an abstraction layer allowing you to load a YOLOv8 model very simply and train it with only the following 3 lines:

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

We chose to train the model over 100 epochs in order to have the most efficient detector possible in terms of efficiency. As shown in the figure, the Mean Average Precision (mAP) at an Intersection over Union (IoU) threshold of 0.5 is 0.891 which means that our face detector based on YOLOv8n performs very well under test conditions with an efficiency of 89.1% real positives.

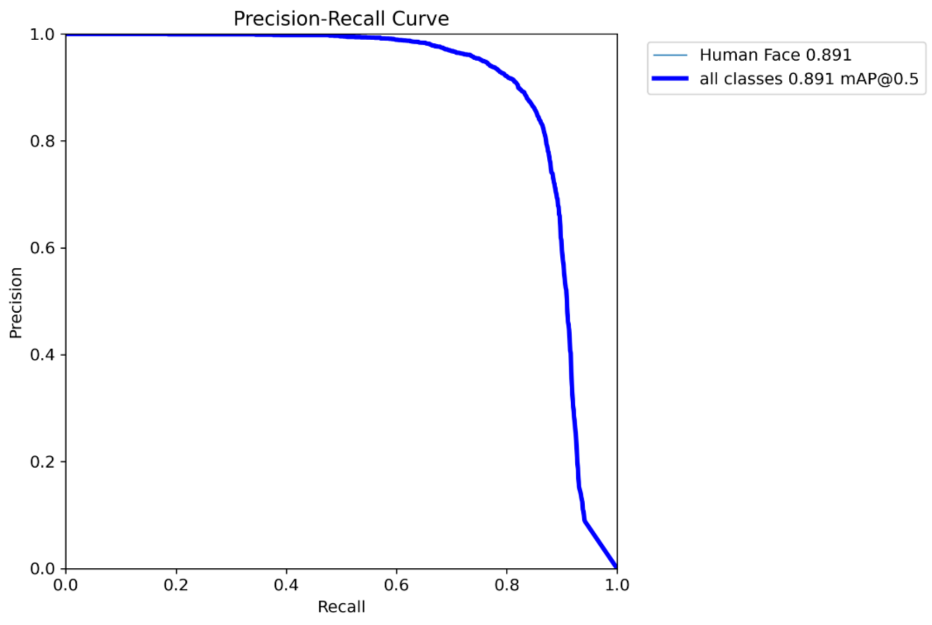


Figure4- Face detector accuracy

4. Implementation

The face detector implementation relies on a FaceDetector class which inherits from the Model abstract class. The Model abstract class defines a generic model with an abstract warm_up method, thus ensuring prior initialization of the model.

In the constructor of the FaceDetector class, the path of the specific YOLO model for face detection that we trained beforehand is provided, and the model is initialized according to this path. The model is then moved to the GPU if available, providing hardware acceleration for detection operations.

The warm_up method is implemented to “warm up” the model by making an initial prediction with an empty image. This loads the model weights and prepares the necessary resources for subsequent inferences. A log message indicates that the YOLO model for face detection is ready for use after this step.

The detect method takes an image as input and performs face detection using the YOLO model. Predictions are extracted from the detection results, and a list of Prediction objects is returned. Each Prediction object encapsulates the coordinates of a bounding box and the confidence associated with face detection.

This implementation offers a robust solution for face detection using YOLO, while providing a modular structure that can be extended to incorporate other features or enhancements specific to project needs.

5. Conclusion

This chapter explored the area of object detectors in detail, with a particular focus on face detection, which is essential for our project. We started by defining object detectors in Deep Learning (DL) and their crucial role in our context. Object detection consists of identifying visual classes in an image, by providing the coordinates of a box containing the object, accompanied by a confidence score.

Historical analysis revealed the significant evolution of object detection over the past two decades. Traditional methods such as the Viola-Jones detector led the way, followed by the revolutionary era of object detection after the advent of Deep Learning. Models like R-CNN, YOLO, and Mask R-CNN marked crucial milestones, leading to the creation of YOLOv8, our choice for face detection.

We opted for YOLOv8-n because of its lightweight and fast speed, which is essential for ensuring real-time performance in our user interface. Model training was carried out with the Face-Detection-Dataset dataset, carefully prepared to meet the specific requirements of YOLO.

In sum, this chapter laid the foundation for face detection in our project, providing an in-depth overview of the state of the art, historical evolution, model choice, training, and considerations. linked to the dataset. These elements form the foundation upon which our ability to effectively integrate face detection into our interactive blurring application for video media and images is built.

VI. Chapter 3: Face comparator

1. Introduction

After successfully designing a high-performance face detector using YOLO, it was time to tackle the most important part of the project: grouping faces into different individuals. This area is known as face recognition in Deep Learning and is a fascinating and growing field.

This chapter will therefore begin with a state of the art of face recognition in order to understand its uses, its limits, as well as the public projects making it possible to determine if two photos represent the same person. Subsequently, the implementation of the FaceComparator class within the project will be detailed then a final part will explain in detail the correction algorithm that we have developed in order to have the most reliable results possible.

2. State of the art of face recognition

A. Presentation

Face recognition is the ability to determine whether two faces are the same or not. In Deep Learning, in order to compare whether two faces belong to the same person, the face images are passed into a model which will act as a feature extractor. This model is generally a CNN because this type of neural network turns out to be very efficient for this type of task. Feature extractors take an image as input and return a vector of numbers representing the features of the input image. To have a functional feature extractor, the model must be trained on two numerous images in order to identify the characteristics of each image as best as possible.

Once the features have been extracted from the two images containing the faces, a distance is calculated between the two respective vectors. There are two commonly used methods to efficiently calculate the distance between these two vectors:

- Euclidean distance: This method measures the so-called geometric distance between two points or two vectors. The shorter the distance, the higher the similarity between the two entries.
- Cosine distance: This method measures the cosine angle between the two input vectors. It is widely used for vectors because it is less sensitive to the magnitude of their data.

Finally, when the distance is calculated, it is compared to a threshold in order to know if the two faces present in the two images are the same. If the distance is greater than the threshold, the faces are considered different. On the other hand, if it is lower than the threshold, the two faces are considered to belong to the same person[5].

B. Face recognition

When searching for libraries and models implementing face recognition to determine if two faces belong to the same person, the Python library face_recognition[6]quickly emerged as a solution to our problem of extracting and comparing the characteristics of faces. This library is very complete and has many features such as:

- Face detection
- Localization of facial features
- Face classification for certain celebrities
- Face recognition

It was therefore decided to implement this library within our FaceComparator class. After carrying out some tests on the performance of the library, two problems were revealed:

- Extremely long inference times: around 6 seconds to extract the features of two faces and compare them. Given the number of faces that can be present on a video filmed in at least 30 Frame Per Second (FPS), it was inconceivable to have such a long inference time.
- Empty feature vectors on some faces detected by the FaceDetector class.

These performance issues led us to continue our research to find a model to quickly extract facial features.

C. DeepFace

The first comparator showed disappointing results in terms of efficiency and inference time. The second face comparator we found during our research is DeepFace[7]. This library is based on the Visual Geometry Group Face (VGG-Face) model, a face description model developed by VGG and the University of Oxford. The creators of DeepFace modified the architecture of the VGG-Face model by removing the softmax layer in order to use it as a face feature extractor. This model is implemented in DeepFace using the famous Keras library.

To compare faces, this library uses VGG-Face to extract the characteristics of each face, then compares the two vectors obtained using the cosine distance calculation method by default. The threshold used to determine whether faces belong to the same person was set at 0.40 by DeepFace.

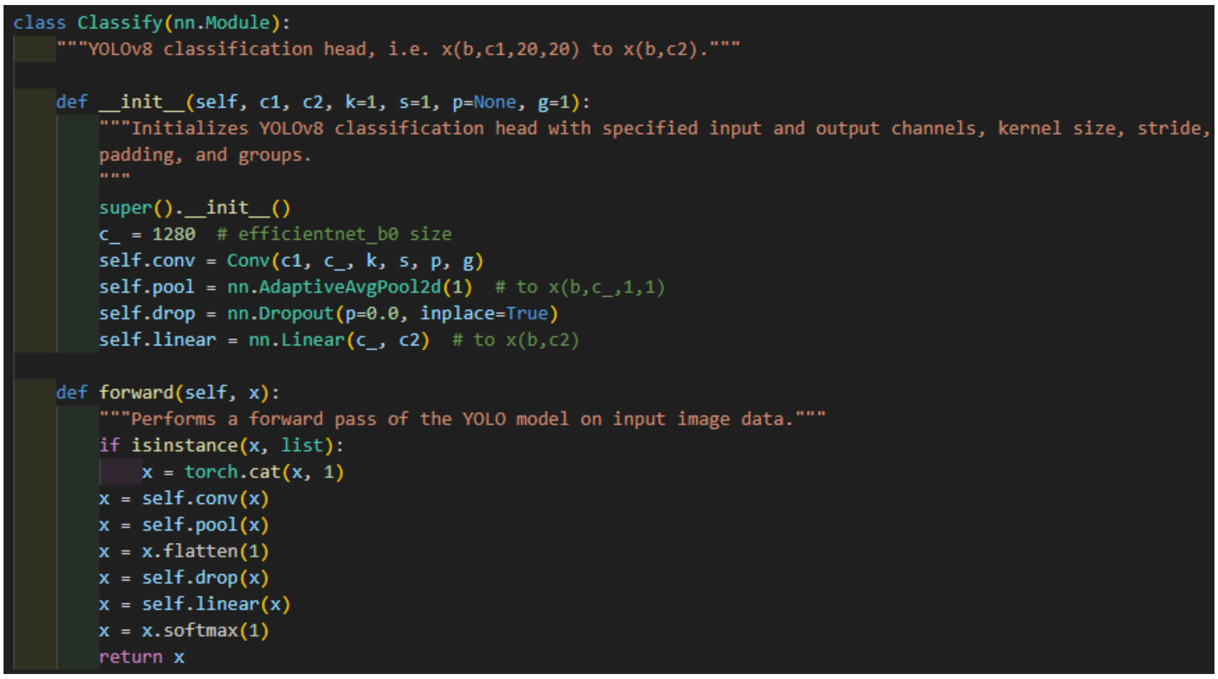
After carrying out a few tests to assess the quality of this library, it appeared that its effectiveness was very satisfactory. Its inference time was much better than that of Face-Recognition but still too slow for our needs (around 150 milliseconds on an NVIDIA RTX 2060) given that the program will have to compare several faces to each image.

D. Yolov8-cls

As we dug deeper, we discovered YOLOv8-cls, a variant of the YOLOv8 object detector designed for image classification. This classifier could be useful to us in order to differentiate the faces of individuals. The only problem being that the model must be trained on a dataset containing the names and faces of people who can be classified. The project does not allow such a dataset because user videos can contain the faces of any person.

After having had a discussion with the researcher and professor Olivier BROUSSE as well as the founder of ultralytics and creator of YOLOv8 Glenn JOCHER[8]on the methods available to us for comparing faces using YOLOv8, the idea came to us to modify the architecture of YOLOv8-cls in order to use it as a feature extractor.

Indeed, the YOLOv8 classifier is composed of a softmax layer as shown in Figure 5.



```
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
```

Figure5- YOLOV8-cls Head

This layer allows the model to normalize the output of the linear layer into a probability distribution representing the probability that the input image has of belonging to each class. By removing the so-called fully connected linear layer which is responsible for mapping the features into the class space and the softmax layer, the model then becomes an ideal feature extractor for performing clustering. The output feature vector is then of size 1280.

We therefore trained the YOLOv8-cls model on 1000 individualsof the VGG-Face dataset each containing between 300 and 500 faces. This training allowed the model to clearly identify the characteristics that dissociate the faces.

Once the feature vectors of two faces were recovered, it was necessary to first choose a threshold which will allow, once the cosine distance has been calculated between the two images, to know whether they correspond to the same faces or not.

The cosine distance between two vectors A and B is given by equation (1):

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

Choosing this threshold is of capital importance in order to determine whether faces belong to a person. For this, we chose to use the same protocol as the creators of DeepFace, documented in the article [5]. This article explains how to calculate the threshold by performing numerous comparisons including true positives, false positives, true negatives as well as false negatives.

After following the tutorial, we unfortunately realized that the 1280 features extracted by the modified YOLOv8-cls model were not precise enough to define whether a face belonged to a person or not. In fact, the distance curves of true positives and false positives were much too close to define a threshold separating the two.

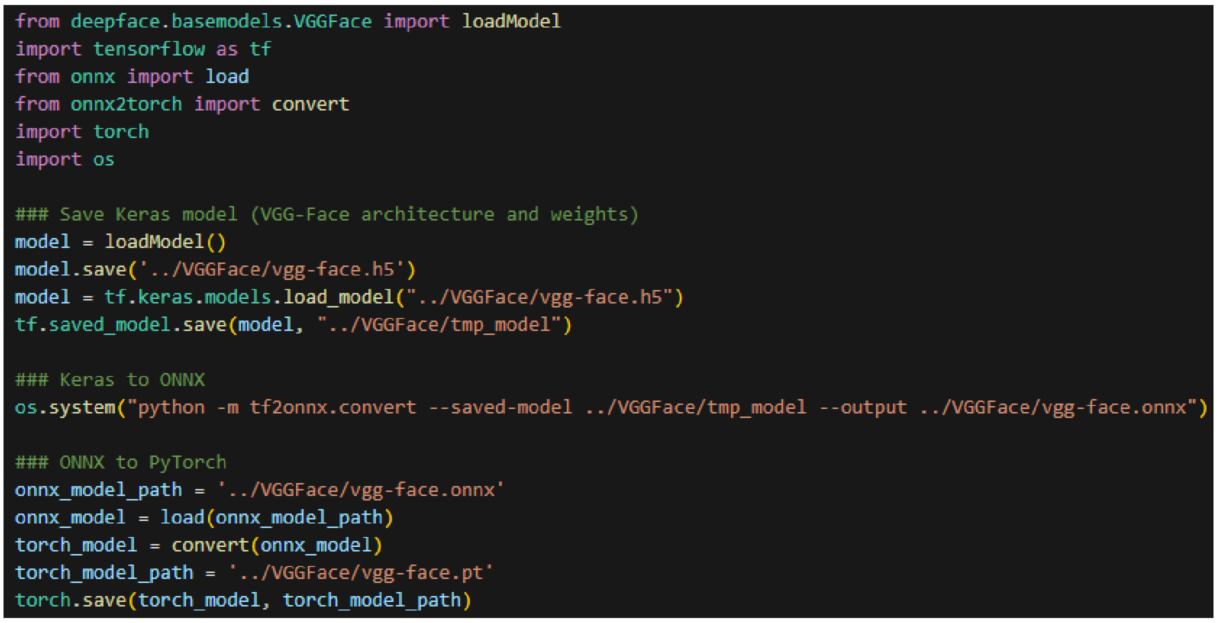
E. VGG-Face

As explained previously, the comparison of faces using the DeepFace library gave very satisfactory results but with an inference time that was too long for our needs. We therefore decided to explore in depth the code used in DeepFace to see if there was any possibility of optimizing certain parts.

By exploring the classes used by DeepFace, it appeared that this library used Keras to implement the VGG-Face model. Keras is a high-level library allowing you to implement Deep Learning networks very quickly. This library provides a layer of abstraction making it accessible to beginners. On the other hand, this library makes calculations on the Graphic Processing Unit (GPU) or graphics card particularly time-consuming due to its abstraction layer.

PyTorch is known for its dynamic computational graph and efficient memory management. This is why we decided to reimplement this model in PyTorch, in order to observe the difference in inference times. Rather than reprogramming the layers of the VGG-Face model one by one using the PyTorch library and retraining the model on the VGG-Face dataset containing 10,000 individuals with a total of 2.6 million images, we decided to 'extract the Keras model from DeepFace. The last two layers of the VGG-Face model are already removed in the DeepFace library, allowing its use as a feature extractor. In addition, DeepFace contains the weights of the network trained on the VGG-Face dataset, thus avoiding us having to retrain it which would have taken several days.

To convert the model to the format used by PyTorch, we went through the Open Neural Network Exchange (ONNX) format. This format makes it possible to represent machine learning models and was developed to allow interoperability between different libraries, each using their own data format. This format was therefore used to move from a Keras model to an ONNX model, then from an ONNX model to a PyTorch model. Figure 6 shows the script used to switch from Keras format to PyTorch format.



```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)
```

Figure6- Converting Keras model to PyTorch

Once the model was converted to PyTorch as well as these weights, the inference time of VGG-Face in this new format was measured and it was quite surprising to observe that the transition from Keras format to PyTorch format reduced the time inference by 80%, going from 151 ms to just 34 ms. Figure 7 illustrates the inference time of VGG-Face on 160 images in Keras, ONNX and PyTorch formats. It clearly appears that the PyTorch library provides excellent results compared to the other two.

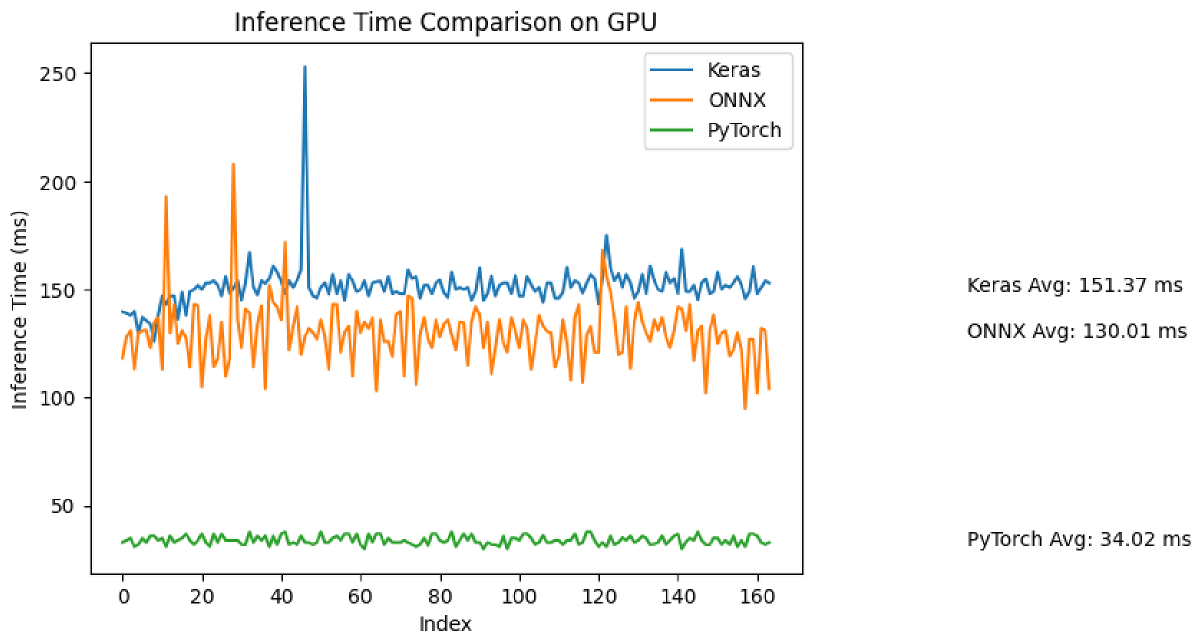
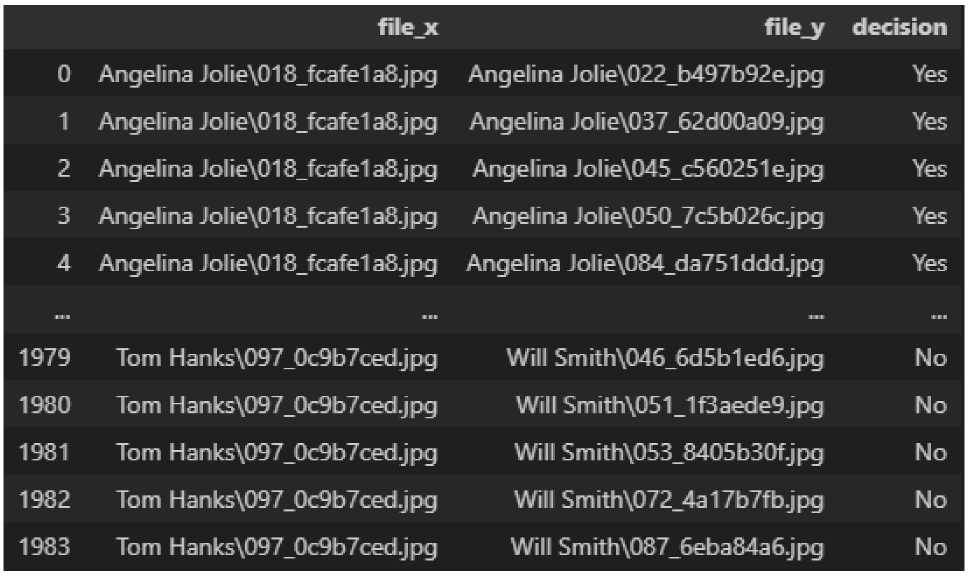


Figure7- Benchmark of VGG-Face inference time in different formats

The inference time of the model was therefore satisfactory in this new format, but it was necessary to ensure that its performance in terms of effectiveness in dissociating individuals was still satisfactory and to recalculate the threshold. To do this, we followed the article [5] explaining how to calculate the threshold by carrying out a certain number of face comparisons. The results of this threshold calculation are available in the tests/classifier_tests.ipyb file.

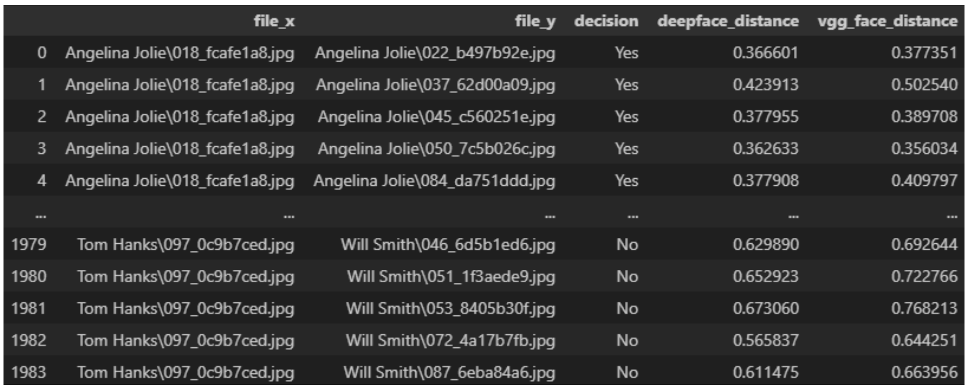
We first selected 17 individuals, each containing several faces. A panda Dataframe was then created to group the comparison results, firstly containing the file_x, file_y and decision columns. These 3 columns respectively indicate the first face to compare, the second face to compare and the expected result. True positives were therefore generated by associating two faces of the same person, and true negatives were generated by associating two faces of different people as shown in 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

Figure8- Generation of true positives

Two columns were then added allowing the comparison distance of the two faces to be recorded for each line with respectively the Keras model of VGG-Face used by DeepFace, and the PyTorch model that we generated. The result of these comparisons is illustrated in 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

Figure9- Comparison results

These comparisons allowed us to obtain the following two graphs presented in Figure 10, illustrating the distribution of distances for each model. The blue curve represents the comparison distances of faces belonging to the same person while the orange curve represents the comparison distances of faces belonging to different people.

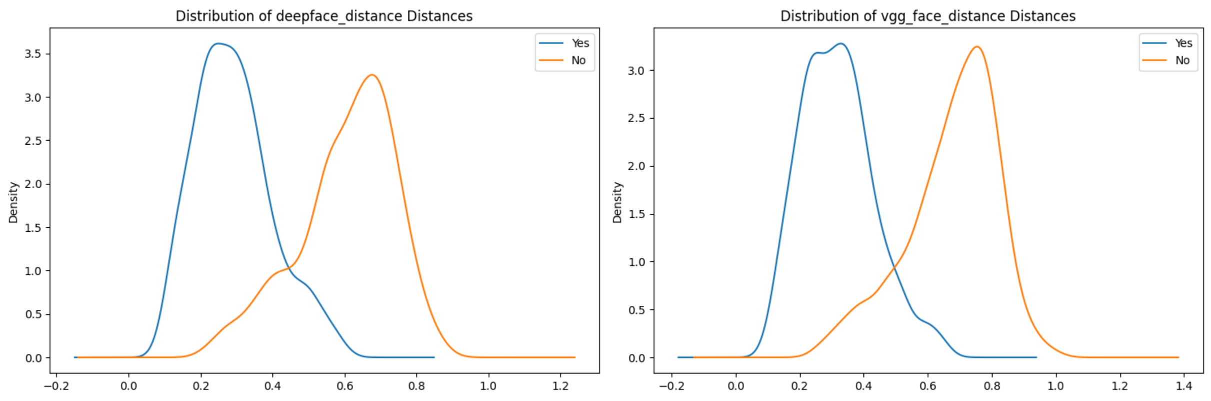


Figure10- Distance distributions of the two models

With the Keras model, a threshold seems to be distinguished around 0.4 while the PyTorch model indicates a separation of the two curves around 5.0. However, these two curves show that this model is not perfect because certain distances are very small even though they are the result of comparisons of faces that do not belong to the same person.

The threshold can be calculated in 3 different ways:

- A decision tree: allows you to find an optimal threshold that best separates comparisons of the same people and different people.
- The 2-sigma rule: means that values are within two standard deviations of the mean in a distribution. It would make it possible to obtain a threshold below which 95.45% of the distances of comparisons of faces of the same person are found.
- The 3-sigma rule: means that values are within 3 standard deviations of the mean in a distribution. It would make it possible to obtain a threshold below which 99.73% of the distances of comparisons of faces of the same person are found.

In order to identify the best threshold, we calculated all 3 of them for the two models and we obtained the confusion matrices represented by Figure 11.

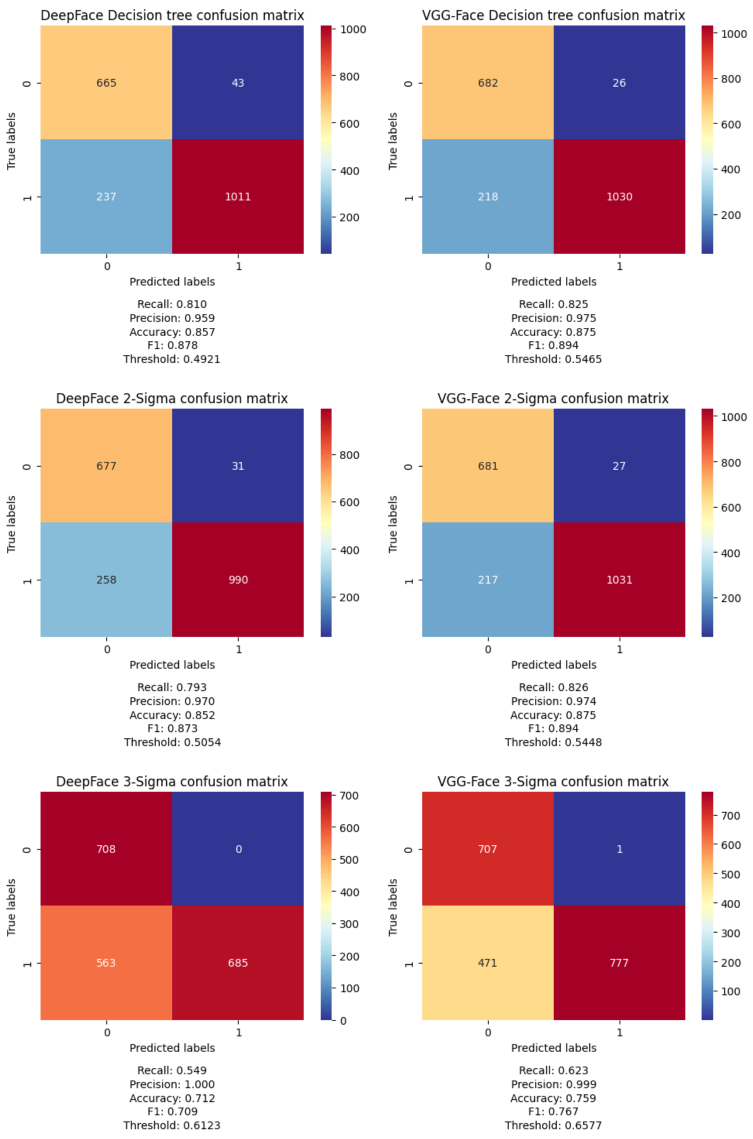


Figure11- Confusion matrices of the 3 thresholds for the 2 models

For the PyTorch model, the thresholds calculated by the decision tree, 2-sigma and 3-sigma methods are 0.55, 0.54 and 0.65 respectively. As shown in Figure 10, the higher the threshold, the greater the number of false positives (faces mistakenly considered to belong to the same person). On the other hand, the lower the threshold, the greater the number of false negatives.

Given the confusion of the two curves in the distance distribution, it was clear that it was not possible to obtain a perfect threshold. It is for this reason that we decided to develop a correction algorithm, which will allow us to reduce the number of false negatives. We therefore chose to use a high threshold within the model which will in the worst case generate false positives which will be corrected subsequently. The comparison algorithm is detailed in part 5 of this chapter.

3. Implementing the FaceComparator

The Face Comparator implementation relies on the VGGFaceComparator class, which inherits both the Model abstract class and the FaceComparator interface. This class uses the pre-trained VGG-Face PyTorch model to extract features from faces and perform comparison based on cosine distance.

In the class constructor, the VGG-Face PyTorch model path is provided, the model is loaded, and the execution device (CPU or GPU) is determined. The model is then moved to the appropriate device (GPU if available). The similarity threshold is set to 0.40, and the warm_up method is called.

The warm_up method takes an empty image, pre-processes it using the image_preprocess method, and makes a prediction with the VGG-Face model. This ensures that the model is initialized and ready for subsequent comparisons.

The image_preprocess method performs the necessary preprocessing on a given image before entering it into the VGG-Face model. This includes resizing the image, converting to float32, and adding an extra dimension.

The compare method is used to compare two faces. The input faces are pre-processed by the image_preprocess method, the features of the faces are extracted using the VGG-Face model, and the cosine distance between these features is calculated. Depending on this distance, the _is_same_person method determines whether or not the faces belong to the same person.

The _findCosineDistance and _is_same_person methods are auxiliary methods to calculate the cosine distance between features of faces and determine whether the faces belong to the same person based on the set threshold, respectively.

This implementation provides a robust approach for face comparison using the VGG-Face model and offers a modular structure that can be extended to new features but also to other face comparators depending on the needs of the project.

4. Analysis of a video frame

The analysis of a video frame takes place in the PersonManager class. The analyze_frame method is called for each frame, which triggers face detection using the FaceDetector. The detected faces are then analyzed and associated with existing people or added as new people.

Analysis of a Video Frame:

The analyze_frame method takes as a parameter the frame index in the video as well as the frame itself. It starts by detecting faces in the frame using the face detector (face_detector.detect). The predictions obtained are used to create instances of the Face class representing each detected face, by storing the frame index, the bounding box of the detected face and its confidence score (Prediction object).

For each detected face, the _save_face method is called to assign the face to a person. The following steps are performed for each face detected:

- Face Save:The _save_face method takes the detected face, the frame and a list of people IDs in the current frame. She crops the face using the image editor (ImageEditor.crop) and gets the features of the cropped face using the face comparator (face_comparator.get_features).
- Checking the Existence of People:If no person has been identified so far (_is_persons_empty), a new person is created and the face is associated with this person as a reference face (cropped_face attribute of the Person class).
- Comparison with Existing People:If people already exist, the method iterates through each person to compare the face with existing faces. The comparison is based on the cropped_face_features and the person's reference facial features. If an existing person matches the current face, the face is added to that person. Otherwise, a new person is created. In the case where the face has been associated with an already existing person, the reference face of this person can be replaced by the current face if the confidence score of the latter is higher.
- Return of the Person Identifier:The identifier of the person (UUID) associated with the face is returned to avoid adding several faces in a frame to the same person. Indeed, in order to improve the performance of the algorithm, we have hypothesized that a person cannot be present twice on the same frame.

Comparison of Faces and Features:

The PersonManager class also exposes methods like compare_faces and compare_features for comparing faces and features respectively. These methods use the face comparator (face_comparator) to make the necessary comparisons.

In summary, the analysis of a video frame relies on face detection, comparison with existing faces, and association of faces to people. Comparisons are made using features extracted from faces using the face comparator. Finally, people are grouped based on these comparisons to improve the consistency of person identification.

5. Correction algorithm

To improve the accuracy of our results for video media, a correction algorithm was developed. This algorithm, once the video has been fully processed (face detection and classification), analyzes the comparison results and applies corrections to minimize false positives.

The algorithm takes place in two main parts: one in person_manager.py and the other in video_processor.py.

person_manager.py:

The PersonManager exposes the group_identical_persons method which iterates over each pair of people and uses the face comparator to determine if two people are the same. If so, they are merged by calling the _merge_persons method.

The _merge_persons method compares the confidence scores of the cropped_face reference faces of the two people. The person with a better confidence score keeps their list of Face objects (as a reminder, Face has an index indicating the image at which the face was detected as well as a Prediction object) and adds the list of Face objects from the another person to his. Then the person with lower trust is removed from the people list, ensuring that each person keeps the most trustworthy face.

video_processor.py:

The correction process begins by calling the _correction method of the VideoProcessor. At this point, the PersonManager has already fully processed the video (face detection and classification) using the face detector and face comparator.

First, grouping identical people is done by calling the group_identical_persons method of PersonManager.

Then, in the main loop, each face detected for each person is considered. For each face, a comparison is made with all the faces of other people. If the comparison indicates that the current face and the other person's face are the same (comparison.is_same_person is true), an additional check is performed.

If the other person's face has a smaller distance from the current face than the current face from the original person's reference face, it means that the other person's face has a better match with the current face that the person originally assigned for that face. In this case, the face is transferred to the other person, and the current face is removed from its original person.

This correction algorithm helps improve the accuracy of person identification by correctly associating detected faces with the people they belong to, while taking into account the confidence associated with each face.

6. Conclusion

This chapter explored facial recognition in detail, with a focus on face comparison methods. We presented popular libraries such as face_recognition, DeepFace or Yolov8-cls. Finally, we integrated into our FaceComparator a VGG-Face model, coming from DeepFace and converted into PyTorch in order to optimize the performance of our application.

Furthermore, the correction algorithm was introduced as a key element to strengthen the reliability of the results. This comprehensive approach, combining a state-of-the-art facial recognition model and personalized optimization, forms the basis of our ability to efficiently group detected faces into distinct individuals.

VII. Chapter 4: User Interface

1. Introduction

The user interface occupies an important place in our project, making it possible to make the link between algorithmic complexity and the end user. In this chapter, we will explore how our solution looks through the integration of Gradio, a powerful Python library that makes it easy to create interactive interfaces for artificial intelligence models.

Thanks to its simplified approach, Gradio offers an intuitive user experience. Our choice of this library stems from its ability to make the power of artificial intelligence accessible without compromising the sophistication of our solution.

We will highlight the different functionalities of the interface, from uploading videos/images to selecting individuals to blur, while highlighting the ergonomic aspects and design choices that optimize the user experience.

2. Interface Overview

Our user interface, designed to be simple and efficient, is divided into two distinct tabs, each dedicated to a specific file type: Video Processing for processing videos and Image Processing for processing images.

Video Processing tab:

When users navigate to this tab, they are greeted with a clean interface that makes it easy to upload a video from their device. The user can view a preview of the video he has just uploaded.

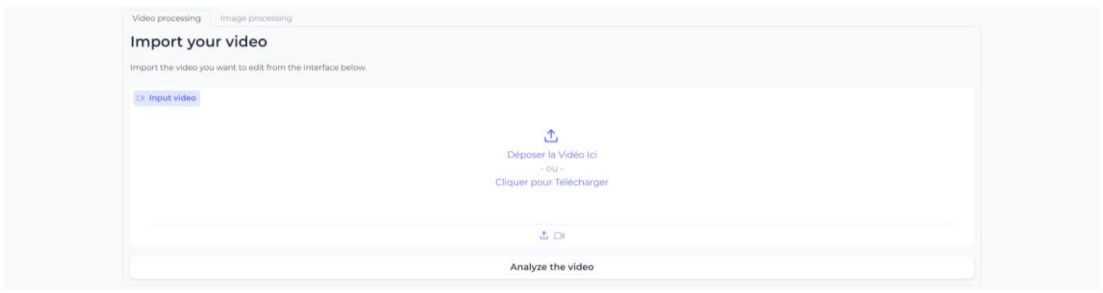
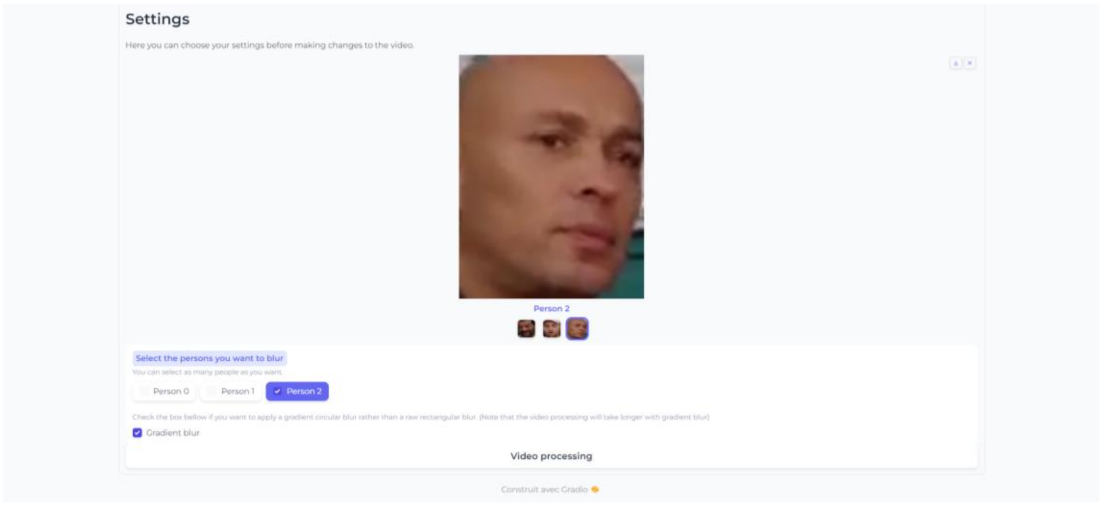


Figure12- "Video processing" tab of the interface (1)

Below this preview, an "Analyze the video" button allows the user to start the face detection process, followed by a progress bar to indicate the progress. The process detects the faces present in the video, groups them according to individuals, and offers an intuitive list of these individuals to the user. For each individual, a checkbox allows or not to anonymize the face of this person in the entire video. A checkbox also allows the user to choose between a "gradient" blur and a raw blur.



Settings

Here you can choose your settings before making changes to the video.

Person 2

Select the persons you want to blur

You can select as many people as you want.

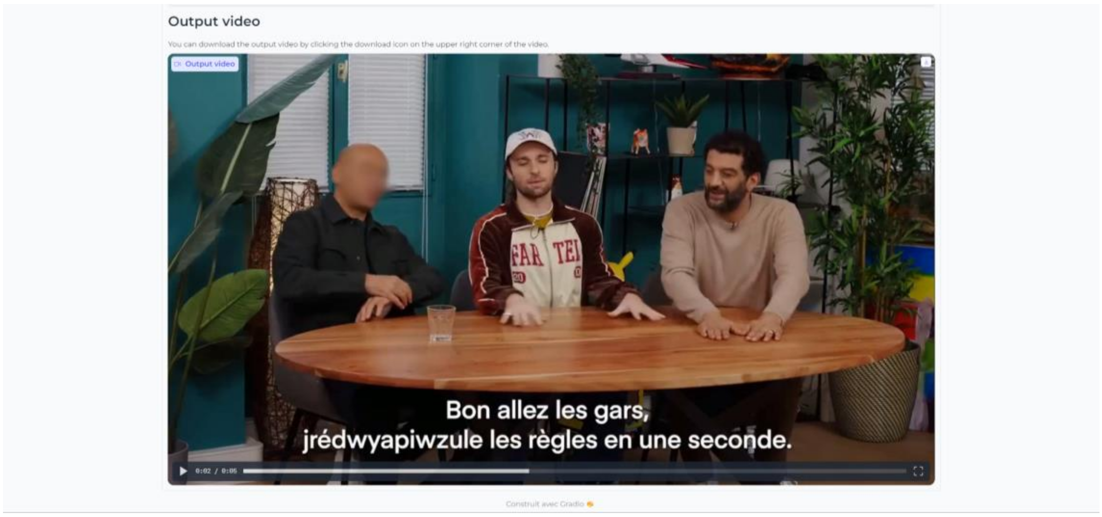
Person 0 · Person 1 · [✓ Person 2]

Check the box bellow if you want to apply a gradient circular blur rather than a raw rectangular blur. (Note that the video processing will take longer with gradient blur)

[✓ Gradient blur]

Figure13- "Video processing" tab of the interface (2)

Once the user's choices have been made, a "Video processing" button launches the process of blurring the selected faces, followed by a progress bar to indicate the progress. When the blurring process is completed, the user can view and download the video result.



Output video

You can download the output video by clicking the download icon on the upper right corner of the video.

Figure14- "Video processing" tab of the interface (3)

Image Processing tab:

This tab is dedicated to image processing. It offers an interface similar to that of Video Processing but optimized for static images. Users can upload an image and, after processing, view the result with the possibility of blurring detected faces identical to the videos interface.

The user-friendly aspect of the interface is manifested by the simplicity of these two tabs. The user is not overwhelmed by complex options, but rather guided intuitively through the processing process, providing a smooth and accessible user experience.

3. Implementation

The implementation of our user interface is based on the use of the Gradio library, chosen for its ease of integration into our project.

Here's how we implemented each tab in our interface:

main.py file:

The main main.py file creates a tabbed interface using Gradio. Two tabs are included, one for Video processing and another for Image processing. The code also uses a "soft" theme (gr.themes.Soft()).

video_interface.py file:

Initializing the Video interface (VideoInterface):

- *__init__:*Initializes the class with a Blocks object from Gradio, a video processor, and an empty list of personsDTO.

Definition of tabs (part1, part2, part3):

- *part1_video_container:*Section to import the video.
- *part2_video_container:*Section for video processing settings.
- *part3_video_container:*Section for video output.

Event management:

- *video_analysis:*Method to analyze video and get detected faces.
- *update_persons_should_be_blurred:*Method for updating people to be blurred.
- *apply_blur:*Method to apply blur to video.
- *reset:*Resets the video processor and personsDTO list.

image_interface.py file:

Initializing the Image interface (ImageInterface):

- *__init__:*Initializes the class with a Gradio Blocks object, an image processor, and an empty list of personsDTO.

Definition of tabs (part1, part2, part3):

- *part1_image_container:*Section to import the image.
- *part2_image_container:*Section for image processing settings.
- *part3_image_container:*Section for image output.

Event management:

- *image_analysis:*Method to analyze the image and get the detected faces.
- *update_persons_should_be_blurred:*Method for updating people to be blurred.
- *apply_blur:*Method for applying blur to the image.
- *reset:*Resets the image processor and personsDTO list.

General remarks :

The management of sections (part2 and part3 for each interface) is based on modifying CSS styles to show or hide these parts depending on user actions.

The video_analysis and image_analysis functions use the respective video and image processors (VideoProcessor and ImageProcessor) to obtain information about people detected in the video or image.

The update_persons_should_be_blurred function modifies the personsDTOs based on the user's selections, and apply_blur calls the blurring function of VideoProcessor or ImageProcessor, passing the personsDTOs as parameters, among others.

4. Conclusion

Concluding our exploration of the user interface, it is evident that the integration of Gradio played a major role in its success.

Thanks to this platform, we were able to design a user-friendly interface, offering users a simplified experience while ensuring efficient management of the face blurring process. The ease of interaction with the program, allowing faces to be downloaded, analyzed and anonymized in just a few clicks, highlights the effectiveness of our solution in achieving our initial objectives.

In summary, this chapter highlights the symbiosis between the usability of our user interface and the implementation of our artificial intelligence solution.

VIII. General conclusion

In summary, our project resulted in the creation of a user-friendly interface allowing the selective blurring of people in images and videos. Using Gradio was crucial to developing an interactive and intuitive interface, offering two distinct tabs for image and video processing.

The modularity of our approach simplified code maintenance and extension. Accurate face detection and the ability to selectively blur people are strong points of our solution, offering users significant flexibility.

Although our project meets the objectives set, future improvements could be considered such as perfecting the face comparator and accelerating the creation of “gradient” blur as well as the process of correcting detected faces for videos. However, overall, our application represents a successful exploration of computer vision and user interaction.

In conclusion, our project offers a functional and attractive solution facilitating the interactive processing of visual media. It responds directly to our basic problem: “How can the development of such technologies (increasingly constant video recordings) coexist with ethical and legal standards? ". Our application then offers a practical approach by reducing the delicate management of image rights to just a few clicks, particularly when it comes to people present without their knowledge in videos or images.

IX. Sources

[1]. 09/17/2017 Robert C. MARTIN– “Clean Architecture”.

[2]. Gaudenz Boesch – “Object Detection in 2024: The Definitive Guide”, written on 04/12/2023. Article available at:https://viso.ai/deep-learning/object-detection/

[3]. Ultralytics – “Ultralytics YOLOv8”, last modification on 12/28/2023. Article available at:https://github.com/ultralytics/ultralytics

[4]. Fares ELMENSHAWII – “Face-Detection-Dataset”, last modified 06/01/2023. Dataset available at:https://www.kaggle.com/datasets/fareselmenshawii/face-detection-dataset/

[5]. Sefik ILKIN SERENGIL – “Fine Tuning the Threshold in Face Recognition”, written on 05/22/2020.
Article available at:https://sefiks.com/2020/05/22/fine-tuning-the-threshold-in-face-recognition/

[6]. Adam GEITGEY – “face_recongition”, last modification on 06/10/2022. Bookstore available at:https://github.com/ageitgey/face_recognition

[7]. Sefik ILKIN SERENGIL – “DeepFace”, last modification on 01/02/2024. Bookstore available at:https://github.com/serengil/deepface

[8]. Glenn JOCHER – “Glenn JOCHER”, last modification on 11/19/2023. GitHub page available at:https://github.com/glenn-jocher