PERIFÉRICOS Y DISPOSITIVOS DE INTERFAZ HUMANA
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Reconocimiento Facial

UNIVERSIDAD
DE GRANADA

José Luis Molina Aguilar y Sergio España Maldonado

8 de mayo de 2023

Curso 2022-2023
Correo : joselu201@correo.ugr.es

Índice

1. Descripción

Vamos a ver dos programas para detectar caras y otro para el reconocimiento de caras, utilizaremos en ambos Opencv, el primer programa usa algoritmo de cascada para detectar caras el segundo utiliza el modulo de face recognition el cual utiliza deep learning. `https://opencv.org/`

2. Detección de caras vs Reconocimiento facial

`https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html`
La detección de caras y el reconocimiento facial son dos términos relacionados pero diferentes en el procesamiento de imágenes y la inteligencia artificial.

La detección de caras se refiere a la capacidad de detectar la presencia de una cara humana en una imagen o video y ubicarla dentro de la imagen.

El reconocimiento facial, por otro lado, se refiere a la capacidad de identificar una persona en función de sus características faciales. El reconocimiento facial se basa en la comparación de las características faciales de una persona con una base de datos previamente almacenada de características faciales de personas conocidas.

Esta técnica se utiliza en muchas aplicaciones, como la seguridad, el control de acceso y la identificación de personas en fotografías o videos.

3. Modelos de cascada

Los modelos de cascada son un tipo de algoritmo de detección de objetos que se utilizan comúnmente para la detección de rostros en imágenes y videos. La detección de cascada se basa en la utilización de una cascada de clasificadores que funcionan en serie, donde cada clasificador toma una decisión sobre la presencia o ausencia de la característica de interés en una determinada región de la imagen.

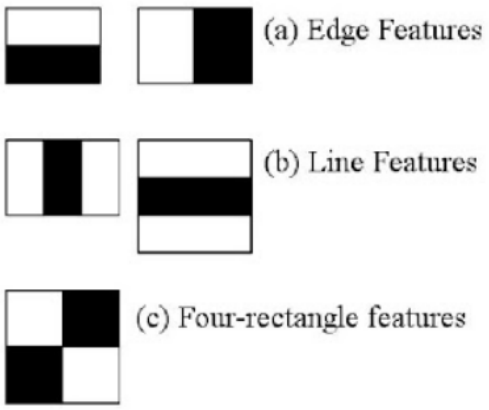

Figura 3.1: Tipo de Característica

Cada característica es un valor único obtenido al restar la suma de píxeles debajo del rectángulo blanco de la suma de píxeles debajo del rectángulo negro.
La ventaja de los modelos de cascada es que pueden ser muy rápidos y eficientes en la detección de objetos en imágenes y videos, lo que los hace útiles para aplicaciones en tiempo real. Sin embargo, también pueden tener limitaciones en la detección de objetos en situaciones variables de iluminación, posición y escala

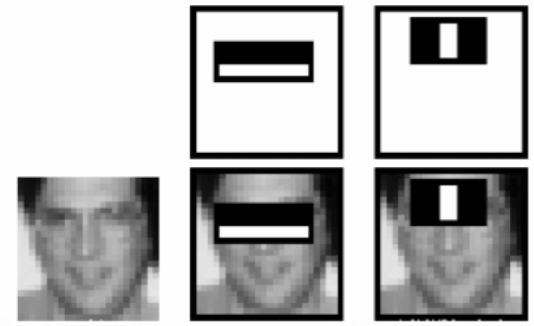

Figura 3.2: Ejemplo de características

La fila superior muestra dos buenas características.
La primera característica seleccionada parece centrarse en la propiedad de que la región de los ojos suele ser más oscura que la región de la nariz y las mejillas.
La segunda característica seleccionada se basa en la propiedad de que los ojos son más oscuros que el puente de la nariz.

Dependiendo de lo que queramos detectar, instanciaremos el clasificador de cascada con los diferentes modelos preentrenados que nos ofrece Opencv.
`https://github.com/opencv/opencv/tree/4.x/data/haarcascades`

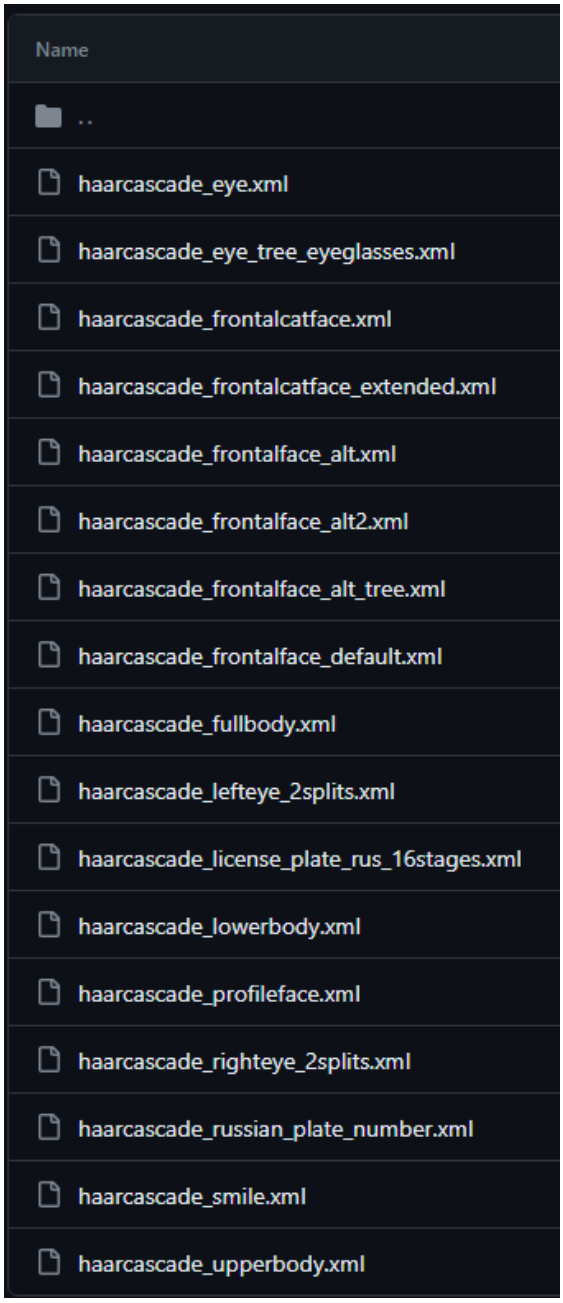

Figura 3.3: Diferentes tipo de clasificadores

Esto lo haremos de la siguiente forma:

```
cascade = cv2.CascadeClassifier("haarcascade<ident>.xml")
```

y ahora simplemente le pediremos que nos identifique de cada frame de la cámara las caras de la siguiente forma:

```
res = cascade.detectMultiScale(frame_gris)
```

es importante destacar que hay que convertir el frame en rgb a escala de grises, esto nos devuelve las coordenadas de la parte superior izquierda de la cara y su ancho y largo
Vemos resultado de este programa.

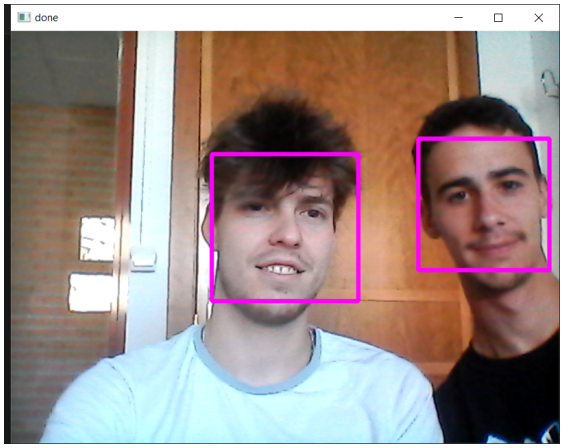

Figura 3.4

4. Python face_recognition

Ahora utilizaremos un modulo de Python llamado face_recognition `https://pypi.org/project/face-recognition/` necesitaremos tener un par de cosas instaladas antes, como cmake, dlib y visual studio.
face-recognition es una herramienta de software libre y de código abierto que se utiliza para el reconocimiento facial y la detección de características en imágenes.

Proporciona una interfaz de programación de aplicaciones (API) fácil de usar para realizar tareas de reconocimiento facial, incluyendo la detección de caras, la extracción de características faciales y la comparación de caras para la identificación de individuos.

Una vez que se han detectado las caras, la biblioteca face-recognition utiliza un algoritmo de extracción de características faciales basado en "dlib"para obtener un vector de características únicas para cada cara detectada.
Estas características incluyen la forma de la cara, las distancias entre los ojos, la nariz y la boca, y otros detalles únicos que se utilizan para identificar a una persona.

Es importante destacar que Opencv utiliza los colores bgr (blue, green, red) en vez de rgb, por esto mismo cada vez que leamos una imagen tendremos que modificar su representación ya que face_recognition si que utiliza rgb.
Utilizaremos la función face_location para localizar las caras de la imagen

```
import face_recognition as face
locs = face.face_locations(framergb)
```

una vez que tenemos esto para poder diferenciar a las personas tendremos que buscar sus característica faciales, eso lo haremos de la siguiente forma.

```
lands = face.face_landmarks(framergb)
```

Ahora tenemos un diccionario en el cual representamos la facción de la cara con su valor. Ahora que tenemos esto, para comparar las caras y determinar la identidad utilizaremos face encodings que devuelve un vector de característica. Estos vectores de características se pueden utilizar para identificar y comparar caras.

```
cod_micara = face.face_encodings(framergb,locs,model='small')[0]
```

Le pasamos la imagen y las posiciones donde hemos localizado las caras, tenemos un tercer parámetro que indica el modelo este se utiliza para especificar el modelo que se utilizará para la detección de caras. Puede ser 'small' para un modelo más rápido pero menos preciso, o 'large' para un modelo más preciso pero más lento.
Ahora simplemente repetiremos estos pasos con cada imagen de la cámara para poder comparar la imagen de referencia con las de la cámara

```
face.compare_faces(cod_micara, cara_cam)
```

Vemos resultado de este programa.

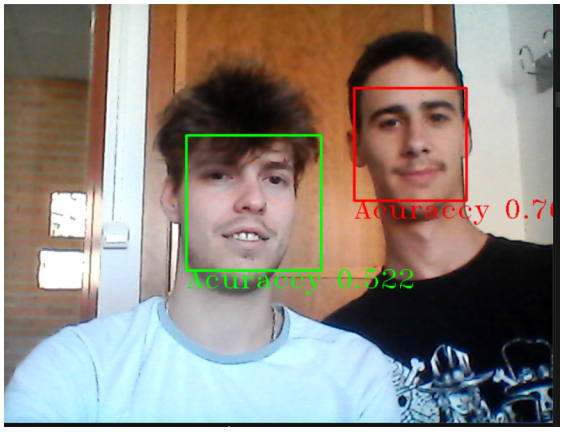

Figura 4.1