

DeepFace on CCTV

Manual del programador

- **NOMBRE Y APELLIDOS:** Ruth Torres Gallego
- **CORREO ELECTRÓNICO:** rutorresg@alumnos.unex.es

ÍNDICE

1. Propósito.....	3
2. Licencia.....	3
3. Instalaciones previas.....	4
4. Guía.....	5
4.1. Organización del repositorio.....	5
4.1.1. Archivo main.py.....	5
4.1.2. Archivo Camera_CCTV.py.....	5
4.1.3. Archivo AnalyzeAllImages.py.....	8
• Función analyze_face.....	9
• Función detect_face_and_hair.....	9
• Función get_dominant_color.....	11
• Función classify_hair_color.....	11
• Función get_hair_color.....	12
• Función detect_glasses.....	13
• Programa principal.....	13
4.1.4. Carpeta functionsSeparated.....	18
4.1.5. Carpeta haarcascades.....	18
4.1.6. Carpeta images.....	18
4.1.7. Carpeta info.....	18
4.1.8. Carpeta captures.....	18
4.1.9. Carpeta intruders.....	18
4.2. Ejecución del proyecto.....	18

1. Propósito

El **objetivo** de este proyecto es crear una mini **cámara de vigilancia**, que sea capaz de detectar rostros, guardar una captura, y analizar dicho rostro. Los rasgos que detecta son: edad aproximada, sexo, raza, emoción, color de pelo y si lleva gafas o no.

Este proyecto está subido en un repositorio público de GitHub, el cual se puede ver a través del siguiente enlace:

<https://github.com/Ruth-Torres/DeepFace-on-CCTV>

2. Licencia

El proyecto usa una licencia MIT, tal y como se indica en el propio [repositorio del proyecto](#).

MIT License

Copyright (c) 2024 Ruth Torres Gallego

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

3. Instalaciones previas

Para poder ejecutar este programa, será necesario instalar algunas librerías:

- **Python** (La versión utilizada para este proyecto es la 3.12.6)
- **OS**: una librería que viene instalada con Python. Proporciona funcionalidad independiente del sistema operativo.
- **Datetime**: una librería que viene instalada con Python y sirve para manejar fechas en Python.
- **Subprocess**: una librería que viene instalada con Python. Es una herramienta que te permite ejecutar otros programas o comandos desde tu código Python.
- **OpenCV**: una librería de computación visual para el procesamiento de imágenes en Python. Se instala con el siguiente comando:

```
pip install opencv-python
```

- **Numpy**: una librería para crear vectores y matrices multidimensionales (arrays) en Python.
IMPORTANTE: para este proyecto se ha usado la versión **1.26.4**, ya que la librería MediaPipe necesita una versión de Numpy anterior a la 2.0.0 y la librería TensorFlow necesita una versión de Numpy entre la 2.1.0 (sin incluir) y la 1.26.0 (incluida). Para instalar Numpy-1.26.4, se ejecuta el siguiente comando:

```
pip install numpy==1.26.4
```

- **DeepFace**: una librería para la detección de caras y el análisis facial en Python.

```
pip install deepface
```

- **TensorFlow**: una librería de Machine Learning desarrollada por Google, necesaria para usar DeepFace.

```
pip install tf-keras
```

- **MediaPipe**: es un marco de trabajo de código abierto desarrollado por Google para construir pipelines de aprendizaje automático multimodales. Es especialmente útil para tareas de visión por computadora, como el reconocimiento facial, el seguimiento de manos, la segmentación de imágenes, entre otros.

```
pip install mediapipe
```

4. Guía

4.1. Organización del repositorio

4.1.1. Archivo `main.py`

El programa principal del proyecto. Ejecuta primero el archivo `Camera_CCTV.py` y una vez termine, ejecutará el archivo `AnalyzeAllImages.py`.

```
import subprocess

# Ruta al primer archivo
primer_archivo = './Camera_CCTV.py'

# Ruta al segundo archivo
segundo_archivo = './AnalyzeAllImages.py'

# Ejecutar el primer archivo y esperar a que termine
subprocess.run(['python', primer_archivo], check=True)

# Ejecutar el segundo archivo después de que el primero haya terminado
subprocess.run(['python', segundo_archivo], check=True)
```

Ya que ejecutar ambos programas en un mismo archivo resultó imposible por la cantidad de errores que aparecían, hubo que separar ambos códigos. Por un lado, el código relacionado con la cámara, y por otro lado, el código relacionado con el análisis facial.

Para ejecutar ambos en una misma ejecución, se utilizó la librería **subprocess**, que permite ejecutar el primer archivo y esperar a que termine. Una vez haya terminado el primero, comenzará la ejecución del segundo.

4.1.2. Archivo `Camera_CCTV.py`

El programa con toda la parte de capturar caras con la cámara. Abrirá la cámara y cada vez que detecte una cara, guardará una captura en la carpeta `captures` con su timestamp correspondiente.

La primera parte del código se limita a importar las librerías necesarias, asegurar de que existe el directorio `captures` para guardar las capturas, inicializar la función de detección facial (y la función para dibujar los trazos para pruebas) de la librería `MediaPipe` y abrir la cámara correctamente.

```

import cv2
import mediapipe as mp
import datetime
import os

# Crear la carpeta 'captures' si no existe
if not os.path.exists('./captures'):
    os.makedirs('./captures')

# Inicializar MediaPipe Face Detection
mp_face_detection = mp.solutions.face_detection
mp_drawing = mp.solutions.drawing_utils

# Configurar la captura de video
camara = cv2.VideoCapture(0)
if not camara.isOpened():
    print("No es posible abrir la cámara")
    exit()

```

La siguiente parte se centra en lo que es la detección facial en sí. Su explicación es la siguiente:

1. Crear el detector de caras:

```

with mp_face_detection.FaceDetection(model_selection=1,
min_detection_confidence=0.5) as face_detection:

```

Se crea un objeto *FaceDetection* de la biblioteca *mediapipe* para detectar caras. *model_selection=1* selecciona un modelo específico y *min_detection_confidence=0.5* establece el umbral mínimo de confianza para considerar una detección como válida.

2. Abrir la cámara y leer los frames:

```

while camara.isOpened():
    ret, frame = camara.read()
    if not ret:
        print("No es posible obtener la imagen")
        break

```

Se abre la cámara y se leen los frames en un bucle. Si no se puede obtener un frame (*ret* es *False*), se imprime un mensaje de error y se sale del bucle.

3. Convertir el frame a RGB:

```
rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

OpenCV lee las imágenes en formato BGR por defecto, por lo que se convierten a RGB, que es el formato que *mediapipe* espera.

4. Detectar caras:

```
results = face_detection.process(rgb_frame)
```

Se procesan los frames convertidos para detectar caras.

5. Procesar las detecciones:

```
if results.detections:
    for detection in results.detections:
        # Dibujar las detecciones en el frame
        # mp_drawing.draw_detection(frame, detection)

        # Guardar la imagen con la cara detectada en la carpeta
        'capturas'

        timestamp =
datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
        cv2.imwrite(f'../captures/capture_{timestamp}.jpg', frame)
```

Si se detectan caras, se itera sobre cada detección. Aunque la línea para dibujar las detecciones está comentada, se guarda una imagen del frame con la cara detectada en la carpeta *captures* con un nombre basado en la fecha y hora actuales.

6. Mostrar el frame con las detecciones:

```
cv2.imshow('Cámara de Vigilancia', frame)
```

Se muestra el frame en una ventana llamada "Cámara de Vigilancia".

7. Salir del bucle si se presiona la tecla 'q':

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

Se espera a que se presione la tecla 'q' para salir del bucle.

8. Liberar la captura de video y cerrar todas las ventanas:

```
camara.release()
cv2.destroyAllWindows()
```

Finalmente, se libera la cámara y se cierran todas las ventanas de OpenCV.

4.1.3. Archivo `AnalyzeAllImages.py`

El programa con toda la parte de análisis facial. Cuenta con funciones auxiliares y un programa principal que las llama. Analizará todas las imágenes en la carpeta *captures* y creará un cartel de **INTRUDER** en la carpeta *intruders* con sus datos.

Al igual que antes, la primera parte del código se limita a importar las librerías necesarias, asegurar de que existen los directorios *captures* e *intruders*, inicializar variables globales, carga los clasificadores Haarcascade e inicializar la función de detección facial (y la función para dibujar los trazos para pruebas) de la librería MediaPipe.

```
import os
import cv2
from deepface import DeepFace
import numpy as np
import mediapipe as mp

# Crear la carpeta 'intruders' si no existe
if not os.path.exists('./intruders'):
    os.makedirs('./intruders')

# Crear la carpeta 'captures' si no existe
if not os.path.exists('./captures'):
    os.makedirs('./captures')

# Propiedades para dibujar en una imagen (rectangulos, círculos, etc.)
color = (0, 255, 255)
grosor = 2

# Cargar los clasificadores Haarcascade para la detección de caras y ojos
face_cascade = cv2.CascadeClassifier('./haarcascades/haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('./haarcascades/haarcascade_eye.xml')
glasses_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
                                         'haarcascade_eye_tree_eyeglasses.xml')

# Inicializar MediaPipe Face Detection
mp_face_detection = mp.solutions.face_detection
mp_drawing = mp.solutions.drawing_utils
```


- Función `analyze_face`

Esta función analiza una imagen para extraer rasgos faciales como la edad, el género, la raza y la emoción usando la biblioteca *DeepFace*. La línea comentada muestra cómo usar *RetinaFace* para una detección más precisa pero más lenta. La función devuelve un diccionario con los resultados del análisis.

```
def analyze_face(image):  
    # Analizar La imagen usando RetinaFace para La detección de caras  
    # analysis = DeepFace.analyze(image, detector_backend='retinaface',  
    actions=['age', 'gender', 'race', 'emotion'], enforce_detection=False)  
  
    # Para un analisis más rápido pero menos preciso, no usar RetinaFace  
    analysis = DeepFace.analyze(image, actions=['age', 'gender', 'race',  
    'emotion'], enforce_detection=False)  
  
    # Extraer Los resultados del análisis facial  
    dicc = analysis[0]  
  
    # Mostrar resultados  
    # print("Resultados del análisis facial:", dicc)  
    # print("Age:", dicc['age'])  
    # print("Gender:", dicc.get('dominant_gender'))  
    # print("Race:", dicc.get('dominant_race'))  
    # print("Emotion:", dicc.get('dominant_emotion'))  
  
    # Devolver Los resultados del análisis facial  
    return dicc
```

- Función `detect_face_and_hair`

Esta función detecta la cara en una imagen y extrae la región del pelo. Utiliza *mediapipe* para la detección de caras y define una región del pelo basada en la posición de la cara detectada. Es una función auxiliar para [get hair color](#).

```

def detect_face_and_hair(image):
    rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    with mp_face_detection.FaceDetection(model_selection=1,
min_detection_confidence=0.5) as face_detection:
        # Detectar caras
        results = face_detection.process(rgb_image)

        # Dibujar Las detecciones en La imagen
        if results.detections:
            for detection in results.detections:
                # Dibujar La detección en La imagen
                # mp_drawing.draw_detection(image, detection)

                # Obtener Las coordenadas de La caja delimitadora
                bboxC = detection.location_data.relative_bounding_box
                ih, iw, _ = image.shape
                x = int(bboxC.xmin * iw)
                y = int(bboxC.ymin * ih)
                ancho = int(bboxC.width * iw)
                alto = int(bboxC.height * ih)

                # Definir La región del pelo (por encima de La frente)
                hair_region_top = max(0, y - int(alto * 0.35)) # Ajustar
según sea necesario
                hair_region_bottom = y-30 # Ajustar según sea necesario
                hair_region_left = x + 30
                hair_region_right = x + ancho - 30

                # Extraer La región del pelo como una nueva imagen
                hair_region = image[hair_region_top:hair_region_bottom,
hair_region_left:hair_region_right]

                # Mostrar La imagen completa
                resized = cv2.resize(image, (0, 0), fx=0.9, fy=0.9)
                cv2.imshow('Imagen completa', resized)

                # Mostrar La imagen con La región del pelo marcada
                cv2.imshow('Hair Region', hair_region)

            return hair_region

```

- Función `get_dominant_color`

Esta función utiliza el algoritmo K-means para encontrar el color dominante en una imagen. Convierte la imagen en un array de píxeles y aplica K-means para identificar los colores más frecuentes. Es una función auxiliar para [get_hair_color](#).

```
def get_dominant_color(image, k=4):
    # Convertir la imagen a un array de píxeles
    pixels = np.float32(image.reshape(-1, 3))

    # Usar K-means para encontrar los colores dominantes
    _, labels, palette = cv2.kmeans(pixels, k, None, (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2), 10, cv2.KMEANS_RANDOM_CENTERS)

    # Contar la frecuencia de cada color
    _, counts = np.unique(labels, return_counts=True)

    # Encontrar el color más frecuente
    dominant_color = palette[np.argmax(counts)]
    return dominant_color
```

- Función `classify_hair_color`

Esta función clasifica el color del pelo basado en valores HSV. Define rangos HSV para diferentes colores de pelo y verifica en qué rango cae el color dominante. Es una función auxiliar para [get_hair_color](#).

```
def classify_hair_color(hsv):
    # Definir los rangos HSV para diferentes colores de pelo
    colors = {
        "castanio claro": [(10, 50, 50), (30, 255, 255)],
        "castanio oscuro": [(10, 50, 20), (30, 255, 100)],
        "castanio": [(10, 50, 50), (20, 255, 150)],
        "negro": [(0, 0, 0), (180, 255, 50)],
        "rubio": [(20, 50, 150), (40, 255, 255)],
        "pelirrojo": [(0, 50, 50), (10, 255, 255)],
        "canoso": [(0, 0, 50), (180, 50, 255)]
    }

    # Verificar en qué rango de color cae el valor HSV dado
    for color_name, (lower_bound, upper_bound) in colors.items():
        if all(lower_bound[i] <= hsv[i] <= upper_bound[i] for i in range(3)):
            return color_name
    return "otro color"
```

- Función `get_hair_color`

Esta función detecta la cara y obtiene la región del pelo (usando [`detect_face_and_hair`](#)), encuentra el color dominante en esa región (usando [`get_dominant_color`](#)) y clasifica el color del pelo (usando [`classify_hair_color`](#)).

```
def get_hair_color(image):
    # Detectar la cara y obtener la región del pelo
    hair_region = detect_face_and_hair(image)

    if hair_region is not None:
        # Encontrar el color dominante en la región del pelo
        dominant_color = get_dominant_color(hair_region)

        # Quitar los decimales de los números
        bgr = [int(num) for num in dominant_color]

        # Convertir el color BGR a HSV
        hsv = cv2.cvtColor(np.uint8([[bgr]]), cv2.COLOR_BGR2HSV)[0][0]

        # Clasificar el color del pelo
        hair_color = classify_hair_color(hsv)

        # Mostrar el color dominante
        # print("Color de pelo (BGR):", hair_color, " ", bgr)

        # Mostrar una imagen con el color del pelo
        img = np.ones((300, 300, 3), np.uint8)*255
        img[:] = bgr # Color RGB
        cv2.imshow('Imagen color de pelo', img)

        # Devolver el color del pelo
        color = hair_color # + " " + str(bgr)
        return color
    else:
        print("No se detectó ninguna cara en la imagen.")
```

- Función `detect_glasses`

Esta función detecta si una persona lleva gafas utilizando cascadas de Haar para detectar caras y ojos. Si se detectan al menos dos ojos, se asume que la persona no lleva gafas. Esto se hace así ya que no es posible detectar gafas directamente con Haarcascades.

```
def detect_glasses(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = image[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)

        """
        # Código para pintar círculos en los ojos
        for (x1, y1, ancho1, alto1) in eyes:
            radio = int((ancho1 + alto1)/4)
            cv2.circle(image, (x1 + x + radio, y1 + y + radio), radio, color,
grosor)
        """

        # Ya que no es posible detectar gafas directamente con Haarcascades,
        # ara detectar gafas habrá que verificar si hay al menos dos ojos por
cara

        if len(eyes) >= 2:
            return "No lleva gafas"
    return "Lleva gafas"
```

- Programa principal

1. Obtener la ruta del directorio actual y de la carpeta *captures*

```
directorio_actual = os.getcwd()
directorio = os.path.join(directorio_actual, 'captures')
```

- `os.getcwd()`: obtiene la ruta del directorio actual donde se está ejecutando el script.
- `os.path.join(directorio_actual, 'captures')`: construye la ruta completa a la carpeta *captures* dentro del directorio actual.

2. Obtener la lista de archivos en el directorio 'captures'

```
archivos = os.listdir(directorio)
```

- **os.listdir(directorio)**: obtiene una lista de todos los archivos y carpetas en el directorio *captures*.

3. Filtrar solo los archivos de imagen (Opcional en este caso)

```
extensiones_validas = ['.jpg', '.jpeg', '.png']
imagenes = [archivo for archivo in archivos if
os.path.splitext(archivo)[1].lower() in extensiones_validas]
```

- **os.path.splitext(archivo)[1].lower()**: obtiene la extensión del archivo y la convierte a minúsculas.

- Se filtran los archivos para incluir sólo aquellos con extensiones **'jpg'**, **'jpeg'**, o **'png'**.

4. Bucle para abrir y analizar cada imagen

```
for imagen in imagenes:
    nombre_archivo, _ = os.path.splitext(imagen) # Obtener el nombre del
archivo sin la extensión
    ruta_imagen = os.path.join(directorio, imagen)
    print(f"Analizando la imagen: {ruta_imagen}") # Imprimir la ruta completa
de la imagen
    img = cv2.imread(ruta_imagen)

    if img is not None:
        try:
            dicc = analyze_face(img)
            hair_color = get_hair_color(img)
            glasses_result = detect_glasses(img)
```

- Se itera sobre cada imagen en la lista **imagenes**.

- **cv2.imread(ruta_imagen)**: carga la imagen desde la ruta especificada.

- Si la imagen se carga correctamente (**img is not None**), se analizan los rasgos faciales, el color del pelo y si la persona lleva gafas usando las funciones [analyze_face](#), [get_hair_color](#) y [detect_glasses](#).

5. Redimensionar la imagen y añadir padding

```
# Redimensionar la imagen
resized = cv2.resize(img, (0, 0), fx=0.5, fy=0.5)

# Definir el tamaño del padding
top = 150
bottom = 300
left = 100
right = 100

# Definir el color del padding (en este caso, blanco)
color = [255, 255, 255]

# Añadir el padding a la imagen
padded_image = cv2.copyMakeBorder(resized, top, bottom, left,
right, cv2.BORDER_CONSTANT, value=color)
```

- `cv2.resize(img, (0, 0), fx=0.5, fy=0.5)`: redimensiona la imagen a la mitad de su tamaño original.
- `cv2.copyMakeBorder`: añade un borde blanco alrededor de la imagen redimensionada.

6. Añadir texto con la información del análisis

```
# Obtener las dimensiones de la imagen con padding
(h, w) = padded_image.shape[:2]
# Obtener el tamaño del texto "INTRUDER"
(text_width, text_height), baseline = cv2.getTextSize("INTRUDER", cv2.FONT_HERSHEY_SIMPLEX, 2, 2)
# Calcular la posición para centrar el texto "INTRUDER" horizontalmente
x_position = (w - text_width) // 2

# Añadir la información de intrusos a la imagen
cv2.putText(padded_image, f"INTRUDER", (x_position, 100), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 0), 2)
cv2.putText(padded_image, f"Age: {dicc['age']}", (100, h-230), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)
cv2.putText(padded_image, f"Gender: {dicc.get('dominant_gender')}", (100, h-190), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (0, 0, 0), 2)
cv2.putText(padded_image, f"Race: {dicc.get('dominant_race')}", (100, h-150), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 0), 2)
cv2.putText(padded_image, f"Emotion: {dicc.get('dominant_emotion')}", (100, h-110),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)
cv2.putText(padded_image, f"Hair Color: {hair_color}", (100, h-70), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0,
0), 2)
cv2.putText(padded_image, f"Glasses: {glasses_result}", (100, h-30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0,
0), 2)
```

- **cv2.putText**: añade texto a la imagen con la información del análisis facial, incluyendo edad, género, raza, emoción, color del pelo y si lleva gafas.

7. Guardar y mostrar la imagen con la información añadida

```
cv2.imwrite(f'./intruders/intruder_info_{nombre_archivo}.jpg', padded_image)
cv2.imshow('Intruder', padded_image)
cv2.waitKey(0) # Esperar a que se presione una tecla para analizar la siguiente imagen
cv2.destroyAllWindows()
```

- **cv2.imwrite**: guarda la imagen con la información añadida en la carpeta `intruders`.
- **cv2.imshow**: muestra la imagen en una ventana.
- **cv2.waitKey(0)**: espera a que se presione una tecla antes de cerrar la ventana con **cv2.destroyAllWindows**.

8. Manejo de errores

```
except Exception as e:
    print(f"Error al analizar la imagen {nombre_archivo}: {e}")
else:
    print(f"No se pudo abrir la imagen: {ruta_imagen}")
```

- Si ocurre un error durante el análisis de la imagen, se captura y se imprime un mensaje de error.
- Si la imagen no se puede abrir, se imprime un mensaje indicando que no se pudo abrir la imagen

4.1.4. Carpeta **functionsSeparated**

En esta carpeta se encuentran las distintas funciones utilizadas en el programa principal por separado.

4.1.5. Carpeta **haarcascades**

Son clasificadores ya entrenados e importados desde el repositorio de OpenCV. En esta carpeta se encuentran los clasificadores de **caras** y **ojos**.

4.1.6. Carpeta **images**

Una serie de imágenes de prueba para las distintas funciones.

4.1.7. Carpeta **info**

Archivos con información adicional sobre las librerías usadas.

4.1.8. Carpeta **captures**

Las imágenes con caras detectadas por la cámara.

4.1.9. Carpeta **intruders**

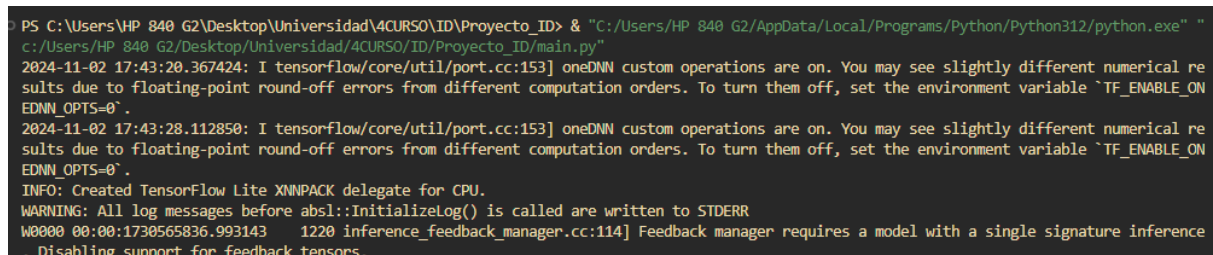
Las imágenes generadas con la descripción de la cara.

4.2. Ejecución del proyecto

Para ejecutar el proyecto, basta con ejecutar el archivo llamado `main.py`, ya sea a través del propio IDE o con el siguiente comando:

```
python main.py
```

Al ejecutarlo, es posible que aparezcan una serie de warnings, pero no impiden la ejecución y funcionamiento del proyecto (Figura 1).



```
PS C:\Users\HP_840_G2\Desktop\Universidad\4CURSO\ID\Proyecto_ID> & "C:/Users/HP_840_G2/AppData/Local/Programs/Python/Python312/python.exe" "
c:/Users/HP_840_G2/Desktop/Universidad/4CURSO/ID/Proyecto_ID/main.py"
2024-11-02 17:43:20.367424: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical re
sults due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ON
EDNN_OPTS=0`.
2024-11-02 17:43:28.112850: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical re
sults due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ON
EDNN_OPTS=0`.
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
W0000 00:00:1730565836.993143 1220 inference_feedback_manager.cc:114] Feedback manager requires a model with a single signature inference
. Disabling support for feedback tensors.
```

Figura 1. Captura de pantalla con warnings iniciales.

Una vez ejecutado, se encenderá la cámara y comenzará a capturar imágenes de las caras detectadas. Las imágenes capturadas se guardarán en la carpeta *captures*. Para cerrar la cámara, habrá que presionar la tecla 'q'.

Luego, empezará a analizar las imágenes capturadas en dicha carpeta. Cada vez que finalice el análisis de una imagen, mostrará los resultados por pantalla. Para pasar a la siguiente imagen, bastará con **presionar cualquiera tecla**.

Cuando ya no haya más imágenes que analizar, el programa terminará. Los resultados del análisis se podrán también ver en la carpeta *intruders*.