

Comparación de Rostros Utilizando Reconocimiento Facial en Python

Autor : Pedro Manuel García Álvarez

06/02/2025

CONTENIDO

1.- Introducción	4
1.1.- Objetivo del Trabajo	4
1.2.- Descripción de las Herramientas y Tecnologías Utilizadas.....	4
1.3.- Instalación del Entorno de Desarrollo	6
1.3.1.- Descarga e instalación de Anaconda	6
1.3.2.- Descargar e instalación de cmake	6
2.- Configuración del Entorno en Anaconda	7
2.1.- Creación de los directorios de entorno de trabajo.	7
2.2.- Creación del entorno virtual	8
2.3.- Activación del entorno de trabajo	8
2.4.- Instalación de las librerías necesarias	9
2.5.- Verificación de las versiones librerías	11
2.6.- Configuración del IDE (Visual Studio Code):	13
2.7.- Instalación de extensiones (Visual Studio Code).....	13
3.- Ejercicio 1: Análisis de Emociones y Difuminado de Rostros	15
3.1.- Objetivo del Ejercicio	15
3.2.- Análisis Detallado del Código	15
3.3.- Resultados y Observaciones	17
4.- Ejercicio 2: Identificación y Difuminado de Rostros Basado en Emociones	19
4.1.- Objetivo del Ejercicio	19
4.2.- Análisis Detallado del Código	19
4.3.- Resultados y Observaciones	23
5.- Ejercicio 3: Comparación de Rostros	24
5.1.- Objetivo del Ejercicio	24
5.2.- Análisis Detallado del Código	25
5.4.- Resultados y Observaciones	28
6.- Pruebas y Ejecución de los Códigos	29
6.1.- Metodología de Pruebas	29
6.1.1.- Objetivos de las Pruebas.....	29
6.1.2.- Tipos de Pruebas Realizadas	30

6.1.3.- Herramientas Utilizadas.....	31
6.1.4.- Casos de Prueba.....	31
6.2.- Resultados de la Ejecución	32
6.2.1.- Resultados de las Pruebas de Unidad	32
6.2.2.- Resultados de las Pruebas Funcionales	32
6.2.3.- Resultados de las Pruebas de Rendimiento.....	33
6.2.4.- Resultados de las Pruebas de Excepciones.....	33
6.2.5.- Resultados de las Pruebas de Integración	33
6.2.6.- Conclusión de los Resultados.....	33
7.- Conclusiones	34
7.1.- Evaluación de los Resultados Obtenidos	34
7.1.1.- Logros Obtenidos	34
7.1.2.- Áreas de Mejora	36
7.1.3.- Aprendizajes y Retos.....	36
8.- Bibliografía	39
8.1.- Documentación oficial de las librerías utilizadas	39
8.2.- Artículos y recursos de referencia	40
8.3.- Otros Recursos.....	41
8.4.- Conclusión de la Bibliografía.....	41
9.- ANEXOS: DESCRIPCIÓN DE TODOS LOS FICHEROS QUE SE ENTREGAN	42
9.1.- Directorio Principal: C:\Procesamiento_Rostros.....	42
9.2.- Directorio C:\Procesamiento_Rostros\Copia_Entorno	42
9.3.- Directorio C:\Procesamiento_Rostros\Exe_Instalación	43
9.4.- Directorio C:\Procesamiento_Rostros\Imagenes.....	43
9.5.- Directorio C:\Procesamiento_Rostros\Source_Ejercicio001.....	44
9.6.- Directorio C:\Procesamiento_Rostros\Source_Ejercicio002.....	44
9.7.- Directorio C:\Procesamiento_Rostros\Source_Ejercicio003.....	44
9.8.- Resumen	44
10.- Instalación del entorno a otro pc.	45
11.- Mapa Mental	46

1.- INTRODUCCIÓN

La presente documentación tiene como propósito detallar el desarrollo de un trabajo basado en el procesamiento de imágenes y el reconocimiento facial mediante códigos en Python. Este trabajo abarca la implementación de diversas funcionalidades como el análisis de emociones, el difuminado de rostros basado en emociones específicas y la comparación de rostros para determinar similitudes. Se utilizará un entorno de desarrollo configurado en Anaconda y Visual Studio Code, aprovechando librerías especializadas en el tratamiento de imágenes y aprendizaje automático.

1.1.- OBJETIVO DEL TRABAJO

El objetivo principal de este trabajo es implementar y analizar soluciones basadas en Python para:

- Detectar y analizar emociones en rostros humanos dentro de imágenes.
- Aplicar difuminado en rostros que presenten emociones específicas como "alegría", "sorpresa" o "miedo".
- Comparar rostros presentes en distintas imágenes para determinar si pertenecen a la misma persona.

A través de estas actividades, se busca:

- Explorar el uso de técnicas modernas de procesamiento de imágenes.
- Integrar librerías avanzadas como OpenCV, DeepFace y face-recognition.
- Evaluar la eficacia de los métodos implementados en diferentes casos de prueba.
- Promover una comprensión práctica de las aplicaciones de la inteligencia artificial y el aprendizaje automático en el ámbito del reconocimiento facial.

1.2.- DESCRIPCIÓN DE LAS HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS

Para el desarrollo de este trabajo se utilizarán diversas herramientas y tecnologías clave:

- **Python:** Lenguaje de programación principal por su versatilidad y amplia gama de librerías para el procesamiento de imágenes y la inteligencia artificial.
- **Anaconda:** Plataforma que facilita la gestión de entornos virtuales y la instalación de paquetes necesarios.
- **Visual Studio Code:** Entorno de desarrollo integrado (IDE) para escribir, depurar y ejecutar el código.

Las librerías utilizadas son:

1. **OpenCV (opencv-python y opencv-python-headless):** Para el procesamiento de imágenes, detección de rostros y manipulaciones visuales como el difuminado.
2. **DeepFace:** Para el análisis de emociones basado en redes neuronales profundas.
3. **face-recognition:** Para la detección y comparación de rostros.
4. **NumPy:** Para operaciones matemáticas y manejo de datos en forma de arreglos.
5. **Pillow:** Para la manipulación de imágenes.
6. **Flask y flask-cors:** En caso de integrar funcionalidades web para visualización de resultados.
7. **scikit-learn:** Para posibles análisis complementarios relacionados con el aprendizaje automático.
8. **Matplotlib:** Para la generación de visualizaciones gráficas.
9. **tqdm:** Para mostrar barras de progreso durante la ejecución de tareas largas.
10. **dlib:** Biblioteca esencial para el reconocimiento facial.
11. **cmake:** Herramienta necesaria para compilar algunas dependencias como dlib.
12. **ffmpeg:** Para manejar contenido multimedia si es requerido.

1.3.- INSTALACIÓN DEL ENTORNO DE DESARROLLO

A continuación, se detalla el proceso de configuración del entorno de desarrollo en Anaconda:

1.3.1.- DESCARGA E INSTALACIÓN DE ANACONDA

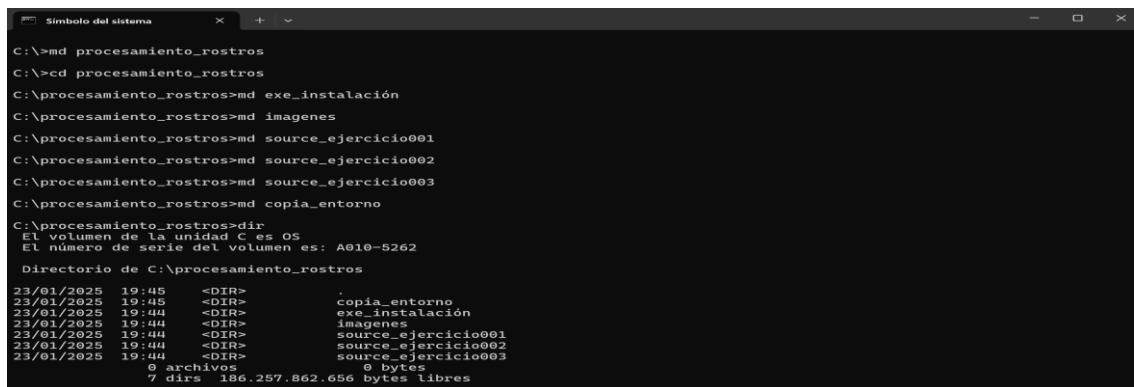
Para instalar Anaconda, descarga el instalador desde el [sitio web oficial](#) de Anaconda y sigue las instrucciones específicas de instalación para tu sistema operativo (Windows, Mac o Linux).

1.3.2.- DESCARGAR E INSTALACIÓN DE CMAKE

Para instalar CMake, descarga el instalador desde el [sitio web oficial](#) de CMake y sigue las instrucciones de instalación específicas para tu sistema operativo (Windows, Linux o macOS).

2.- CONFIGURACIÓN DEL ENTORNO EN ANACONDA

2.1.- CREACIÓN DE LOS DIRECTORIOS DE ENTORNO DE TRABAJO.



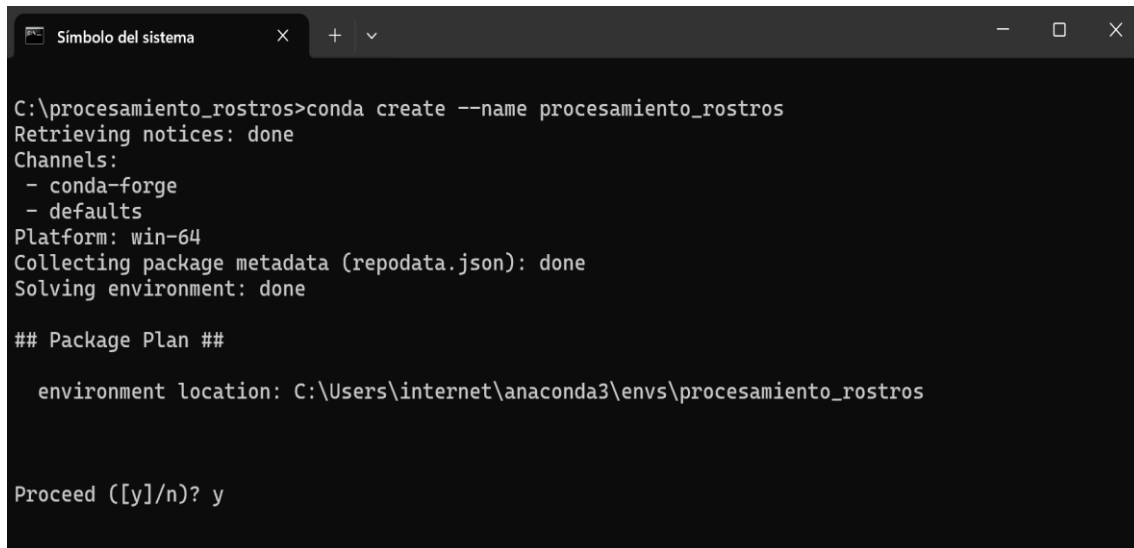
```
Simbolo del sistema
C:\>md procesamiento_rostros
C:\>cd procesamiento_rostros
C:\procesamiento_rostros>md exe_instalación
C:\procesamiento_rostros>md imagenes
C:\procesamiento_rostros>md source_ejercicio001
C:\procesamiento_rostros>md source_ejercicio002
C:\procesamiento_rostros>md source_ejercicio003
C:\procesamiento_rostros>md copia_entorno
C:\procesamiento_rostros>dir
El volumen de la unidad C es OS
El número de serie del volumen es: A010-5262
Directorio de C:\procesamiento_rostros
23/01/2025 19:48 <DIR> .
23/01/2025 19:48 <DIR> copia_entorno
23/01/2025 19:44 <DIR> exe_instalación
23/01/2025 19:44 <DIR> imagenes
23/01/2025 19:44 <DIR> source_ejercicio001
23/01/2025 19:44 <DIR> source_ejercicio002
23/01/2025 19:44 <DIR> source_ejercicio003
0 archivos 0 bytes
7 dirs 186.257.862.656 bytes libres
```

En esta pantalla, se procede a la generación de la estructura de directorios del entorno mediante la siguiente secuencia de comandos: inicialmente, se crea el directorio raíz denominado "procesamiento_rostros" utilizando la instrucción "md". Posteriormente, se navega a este directorio principal empleando el comando "cd". A continuación, se establecen los subdirectorios esenciales, incluyendo "copia_entorno", "exe_instalación", "imágenes", y tres directorios fuente denominados "source_ejercicio001", "source_ejercicio002" y "source_ejercicio003", todos ellos creados mediante la utilización iterativa del comando "md".

En resumen, el entorno de trabajo propuesto para el trabajo de procesamiento de imágenes faciales presenta una estructura de directorios metódicamente organizada, con un directorio principal "procesamiento_rostros" que contiene subdirectorios estratégicamente nombrados para respaldar configuraciones, almacenar ejecutables de instalación, gestionar recursos gráficos y albergar código fuente en diferentes fases o versiones, facilitando así una gestión eficiente y ordenada del trabajo.

2.2.- CREACIÓN DEL ENTORNO VIRTUAL

Abrir la terminal de Anaconda o un terminal compatible. Ejecutar el siguiente comando para crear un entorno llamado "procesamiento_rostros":



```
C:\procesamiento_rostros>conda create --name procesamiento_rostros
Retrieving notices: done
Channels:
 - conda-forge
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

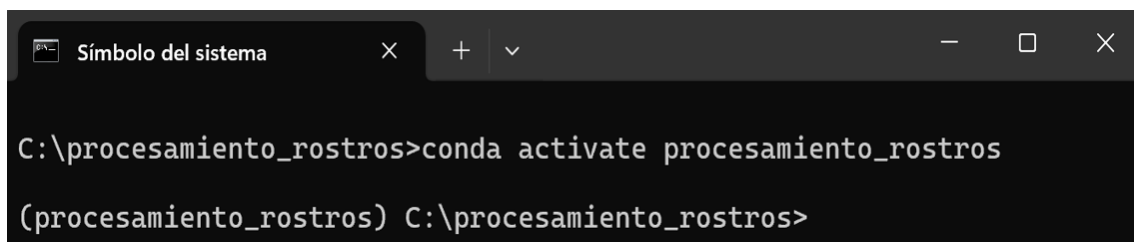
## Package Plan ##

  environment location: C:\Users\internet\anaconda3\envs\procesamiento_rostros

Proceed ([y]/n)? y
```

En esta pantalla, se procede a iniciar la generación del entorno virtual mediante la ejecución del comando "**conda create --name procesamiento_rostros**", tras lo cual se confirma la acción presionando la tecla "Enter" y, seguidamente, se valida la instalación del entorno de trabajo con la pulsación de la tecla "y".

2.3.- ACTIVACIÓN DEL ENTORNO DE TRABAJO



```
C:\procesamiento_rostros>conda activate procesamiento_rostros
(procesamiento_rostros) C:\procesamiento_rostros>
```

En esta pantalla, se ejecuta el comando "**conda activate procesamiento_rostros**" para inicializar y habilitar el entorno virtual previamente configurado, permitiendo el acceso al ambiente de trabajo específico para el trabajo de procesamiento de rostros.

2.4.- INSTALACIÓN DE LAS LIBRERÍAS NECESARIAS

```
Símbolo del sistema - conda x + v - □ x

(procesamiento_rostros) C:\procesamiento_rostros>conda install -c conda-forge dlib
Channels:
- conda-forge
- defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: C:\Users\internet\anaconda3\envs\procesamiento_rostros
added / updated specs:
```

En esta pantalla, se inicia la instalación de la biblioteca *dlib* mediante el comando "**conda install -c conda-forge dlib**", seguido de la pulsación de la tecla Enter para ejecutar el comando, y posteriormente se confirma el proceso pulsando la tecla "y" para comenzar la descarga e instalación de la librería.

```
Símbolo del sistema - conda x + v - □ x

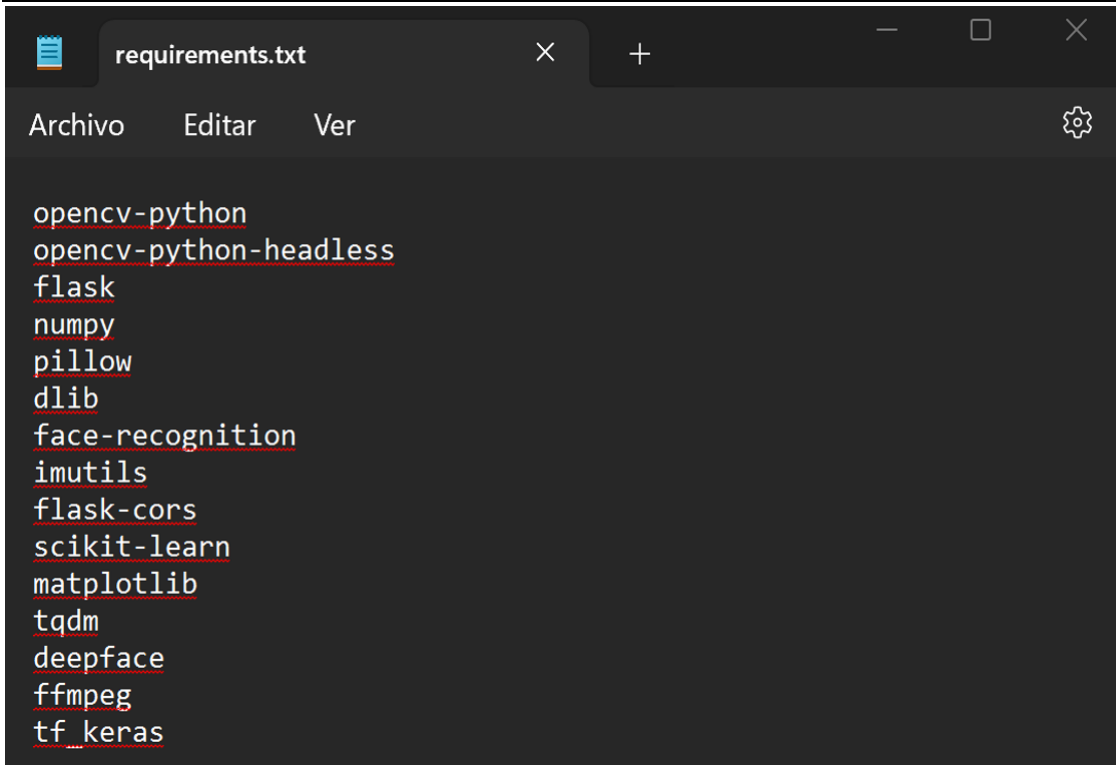
(procesamiento_rostros) C:\procesamiento_rostros>conda install -c conda-forge cmake
Channels:
- conda-forge
- defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: C:\Users\internet\anaconda3\envs\procesamiento_rostros
added / updated specs:
```

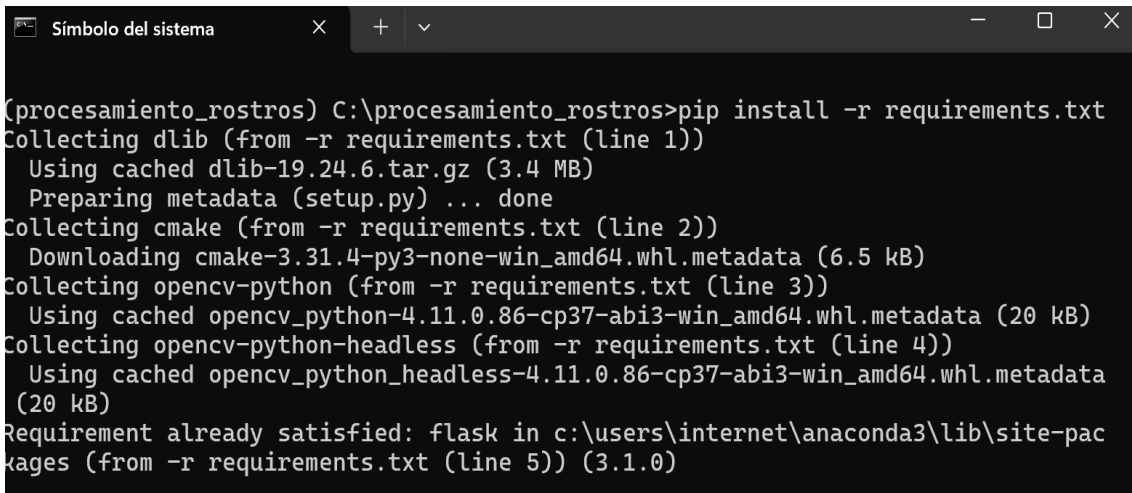
En esta pantalla, se inicia la instalación de la biblioteca *cmake* mediante el comando "**conda install -c conda-forge cmake**", seguido de la pulsación de la tecla Enter para ejecutar el comando, y posteriormente se confirma el proceso pulsando la tecla "y" para comenzar la descarga e instalación de la librería.

En resumen las bibliotecas *dlib* y *cmake* se instalaron individualmente mediante *conda* en lugar de incluirse en el fichero *requirements.txt* porque la instalación por lote presentaba problemas, lo que requirió su instalación manual para garantizar una correcta configuración del entorno.

A screenshot of a text editor window titled 'requirements.txt'. The window has a menu bar with 'Archivo', 'Editar', and 'Ver'. The text inside the editor lists the following dependencies: opencv-python, opencv-python-headless, flask, numpy, pillow, dlib, face-recognition, imutils, flask-cors, scikit-learn, matplotlib, tqdm, deepface, ffmpeg, and tf_keras. Each dependency is on a new line and is underlined in red.

```
opencv-python
opencv-python-headless
flask
numpy
pillow
dlib
face-recognition
imutils
flask-cors
scikit-learn
matplotlib
tqdm
deepface
ffmpeg
tf_keras
```

En esta pantalla, se procede a generar el archivo "requirements.txt" como repositorio documental que compilará de manera ordenada todas las bibliotecas y dependencias necesarias para el trabajo, facilitando su instalación mediante un proceso de configuración por lotes.

A screenshot of a Windows command prompt window titled 'Símbolo del sistema'. The prompt shows the execution of the command 'pip install -r requirements.txt' in the directory 'C:\procesamiento_rostros'. The output shows the installation progress for several packages: dlib (3.4 MB), cmake (6.5 kB), opencv-python (20 kB), and opencv-python-headless (20 kB). It also shows that flask is already installed (3.1.0).

```
(procesamiento_rostros) C:\procesamiento_rostros>pip install -r requirements.txt
Collecting dlib (from -r requirements.txt (line 1))
  Using cached dlib-19.24.6.tar.gz (3.4 MB)
  Preparing metadata (setup.py) ... done
Collecting cmake (from -r requirements.txt (line 2))
  Downloading cmake-3.31.4-py3-none-win_amd64.whl.metadata (6.5 kB)
Collecting opencv-python (from -r requirements.txt (line 3))
  Using cached opencv_python-4.11.0.86-cp37-abi3-win_amd64.whl.metadata (20 kB)
Collecting opencv-python-headless (from -r requirements.txt (line 4))
  Using cached opencv_python_headless-4.11.0.86-cp37-abi3-win_amd64.whl.metadata (20 kB)
Requirement already satisfied: flask in c:\users\internet\anaconda3\lib\site-packages (from -r requirements.txt (line 5)) (3.1.0)
```

En esta pantalla, se procede a la instalación masiva de las bibliotecas especificadas en el archivo "requirements.txt" mediante la ejecución del comando "pip install -r requirements.txt", seguida de la pulsación de la tecla Intro, iniciando así la adquisición e integración automática de todas las dependencias necesarias para el trabajo en el entorno de desarrollo actual.

2.5.- VERIFICACIÓN DE LAS VERSIONES LIBRERÍAS

```
# Programa para verificar version de las librerías

# Importar las bibliotecas necesarias
import subprocess
import sys
```

Este código importa las bibliotecas subprocess y sys, que son necesarias para ejecutar comandos del sistema y acceder a información del intérprete de Python, respectivamente, como parte de un programa diseñado para verificar las versiones de las librerías instaladas.

```
def get_installed_version(package):
    # Intenta obtener la versión con pip
    try:
        # Ejecuta el comando 'pip show' y captura la salida
        output = subprocess.check_output([sys.executable, '-m', 'pip', 'show', package])
        # Decodifica la salida y busca la línea que comienza con 'Version:'
        for line in output.decode('utf-8').split('\n'):
            if line.startswith('Version:'):
                # Extrae y devuelve la versión
                return line.split(':')[1].strip()
    except subprocess.CalledProcessError:
        # Si hay un error (por ejemplo, el paquete no está instalado), pasa al siguiente método
        pass

    # Si no se encuentra con pip, intenta con conda
    try:
        # Ejecuta el comando 'conda list' y captura la salida
        output = subprocess.check_output(['conda', 'list', package])
        lines = output.decode('utf-8').split('\n')
        # Salta las primeras dos líneas de encabezado y busca el paquete
        for line in lines[2:]:
            parts = line.split()
            if parts and parts[0] == package:
                # Si encuentra el paquete, devuelve la versión
                return parts[1]
    except subprocess.CalledProcessError:
        # Si hay un error (por ejemplo, conda no está instalado), pasa
        pass

    # Si no se encuentra la versión, devuelve None
    return None
```

La función `get_installed_version` intenta obtener la versión instalada de un paquete Python utilizando primero pip y luego conda, devolviendo la versión si la encuentra o `none` si no está instalado o no se puede determinar la versión.

```
def check_requirements(file_path):
    print("Librería\t\tVersión Instalada")
    print("-----")
    # Abre y lee el archivo de requisitos
    with open(file_path, 'r') as file:
        for line in file:
            # Extrae el nombre del paquete (asume formato 'paquete==versión' o solo 'paquete')
            package = line.strip().split('==')[0]
            # Obtiene la versión instalada del paquete
            version = get_installed_version(package)
            # Imprime el resultado
            if version:
                print(f"{package:<20}\t{version}")
            else:
                print(f"{package:<20}\tNo instalada")
```

La función `check_requirements` lee un archivo de requisitos, verifica la versión instalada de cada paquete listado, utilizando la función `get_installed_version`, y muestra en la consola el nombre de cada paquete junto con su versión instalada o "No instalada" si no se encuentra.

Librería	Versión Instalada

cmake	3.31.4
dlib	19.24.2
opencv-python	4.11.0.86
opencv-python-headless	4.11.0.86
flask	3.1.0
numpy	1.26.4
pillow	11.1.0
dlib	19.24.2
face-recognition	1.3.0
imutils	0.5.4
flask-cors	5.0.0
scikit-learn	1.6.1
matplotlib	3.10.0
tqdm	4.67.1
deepface	0.0.93
ffmpeg	1.4
tf_keras	2.18.0

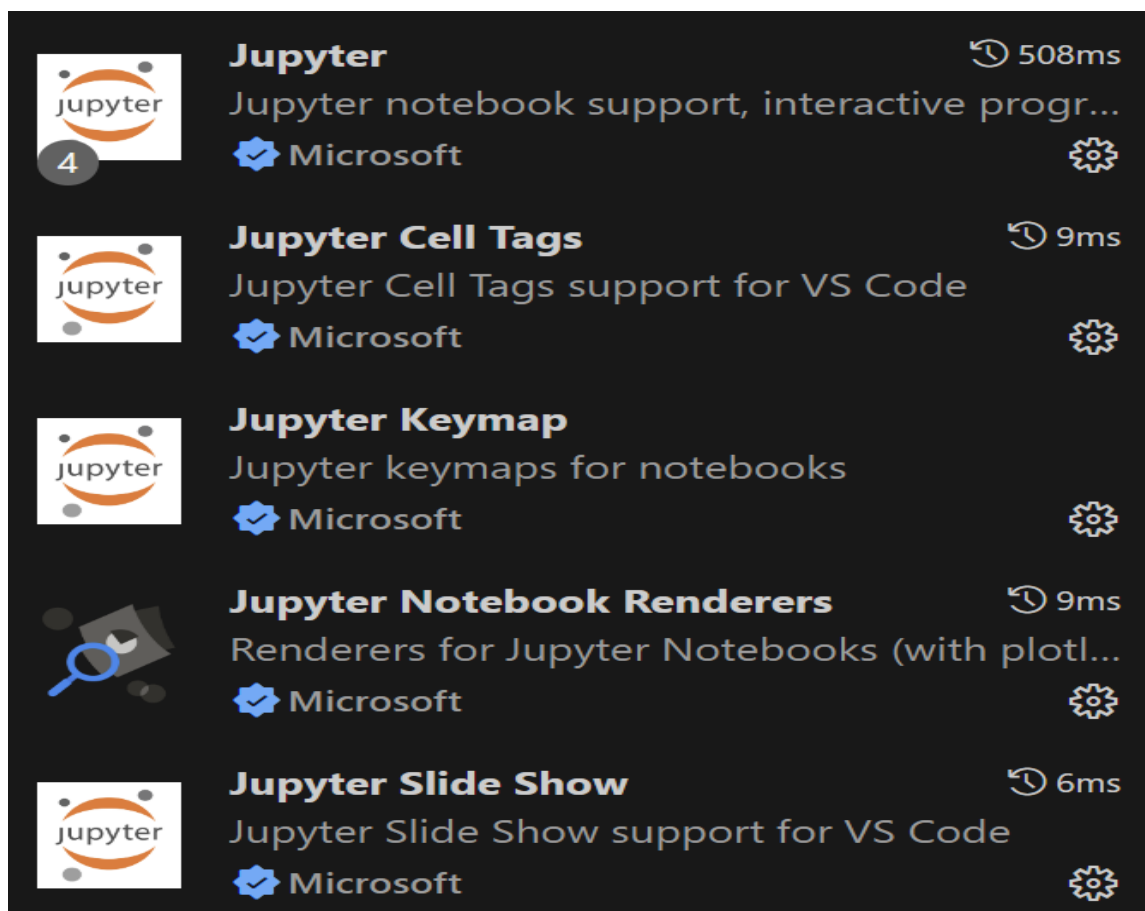
Este resultado muestra una lista de librerías Python junto con sus versiones instaladas, presentando el nombre de cada librería en la

columna izquierda y su correspondiente versión en la columna derecha, o "No instalada" si la librería no está presente en el sistema.

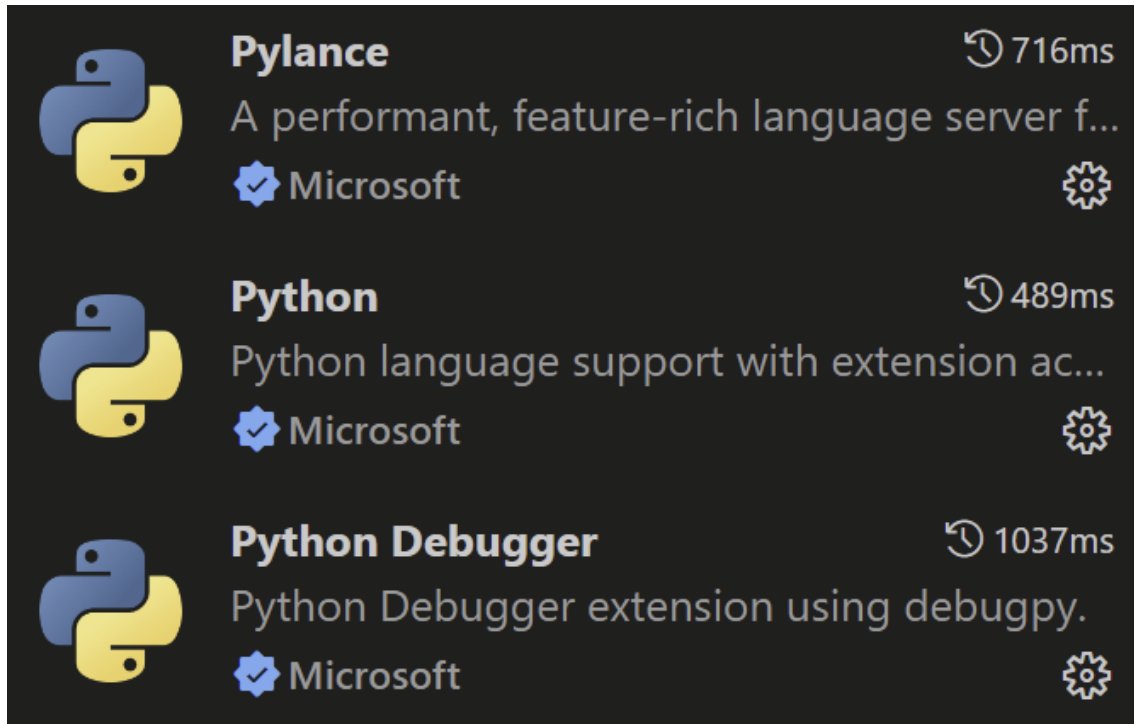
2.6.- CONFIGURACIÓN DEL IDE (VISUAL STUDIO CODE):

Visual Studio Code es un entorno de desarrollo integrado (IDE) desarrollado por Microsoft, cuya instalación implica descargar el instalador desde su [sitio web oficial](#), seleccionar la versión correspondiente al sistema operativo, ejecutar el archivo, aceptar los términos de licencia y seguir los pasos del asistente de instalación para completar la configuración.

2.7.- INSTALACIÓN DE EXTENSIONES (VISUAL STUDIO CODE)



En esta imagen se muestra la instalación de extensiones relacionadas con Jupyter para Visual Studio Code, desarrolladas por Microsoft, las cuales deben estar instaladas para proporcionar soporte a notebooks interactivos, etiquetas de celdas, mapas de teclas, renderización de gráficos y creación de presentaciones tipo slide show.



La imagen muestra extensiones de Microsoft que deben estar instaladas para trabajar con Python en Visual Studio Code, incluyendo Pylance (un servidor de lenguaje eficiente y con múltiples funciones), Python (soporte para el lenguaje Python) y Python Debugger (una extensión para depuración con debugpy).

3.- EJERCICIO 1: ANÁLISIS DE EMOCIONES Y DIFUMINADO DE ROSTROS

3.1.- OBJETIVO DEL EJERCICIO

El objetivo de este ejercicio es implementar un sistema que analice las emociones presentes en los rostros detectados en una imagen y, específicamente, aplique un difuminado a aquellos rostros cuya emoción predominante sea la "alegría". Esto tiene aplicaciones prácticas en la preservación de la privacidad y en el análisis de imágenes para estudios de comportamiento humano.

3.2.- ANÁLISIS DETALLADO DEL CÓDIGO

```
# Ejercicio 1: Reconocimiento Facial, Difuminar Rostros y Análisis de Emociones
#
# Descripción:
# Este script utiliza varias bibliotecas para realizar las siguientes tareas:
# 1. Reconocimiento facial: Detecta rostros en una imagen o un video utilizando la librería face_recognition.
# 2. Difuminado de rostros: Aplica un efecto de desenfoque a las áreas de los rostros detectados en el contenido.
# 3. Análisis de emociones: Emplea la librería DeepFace para identificar la emoción dominante en cada rostro detectado.
# 4. Visualización y exportación: Muestra el resultado del procesamiento y guarda las imágenes modificadas.
#
# Herramientas empleadas:
# - OpenCV: Procesamiento y manipulación de imágenes y videos.
# - DeepFace: Análisis de emociones y atributos faciales.
# - face_recognition: Localización y reconocimiento de rostros en imágenes.
# - NumPy: Manejo de matrices y operaciones numéricas.
#
# Este programa tiene como objetivo demostrar la integración de varias tecnologías
# para el análisis visual y la manipulación dinámica de contenidos multimedia.
```

En esta pantalla, se explica con detalle el código del programa, comentando qué hace cada parte y qué herramientas (librerías) se usarán. Así, se ofrece una descripción clara del propósito del programa y de los recursos que necesita para funcionar.

```
# Importar las bibliotecas necesarias
import cv2 # Librería OpenCV para el procesamiento de imágenes
import numpy as np # Librería NumPy para el manejo de arreglos y matrices
from deepface import DeepFace # Librería DeepFace para el análisis facial y reconocimiento de emociones
import face_recognition # Librería face_recognition para el reconocimiento facial
```

Este código importa bibliotecas esenciales para el procesamiento de imágenes (OpenCV), manejo de datos numéricos (NumPy), análisis facial y reconocimiento de emociones (DeepFace), y reconocimiento facial (face_recognition), estableciendo así las bases para un sistema de visión por computadora capaz de detectar, analizar y reconocer rostros y emociones en imágenes o videos.

```
# Cargar la imagen
imagen = cv2.imread('../imagenes\EJ01_rostros.jpg')
```

Esta línea de código carga una foto llamada "EJ01_rostros.jpg" desde una carpeta de imágenes, para que el programa pueda trabajar con ella.

```
# Detectar los rostros
rostros_localizaciones = face_recognition.face_locations(imagen)
```

Esta línea busca y encuentra las caras en la foto que cargamos antes, marcando dónde está cada una.


```
# Verificar si se detectaron rostros
if not rostros_localizaciones:
    print("No se detectaron rostros en la imagen.")
else:
    # Identificar emociones y difuminar rostros con "alegría"
    for rostro_localizacion in rostros_localizaciones:
        top, right, bottom, left = rostro_localizacion
        rostro = imagen[top:bottom, left:right]

        # Análisis de emociones
        try:
            resultado = DeepFace.analyze(rostro, actions=['emotion'], enforce_detection=False)

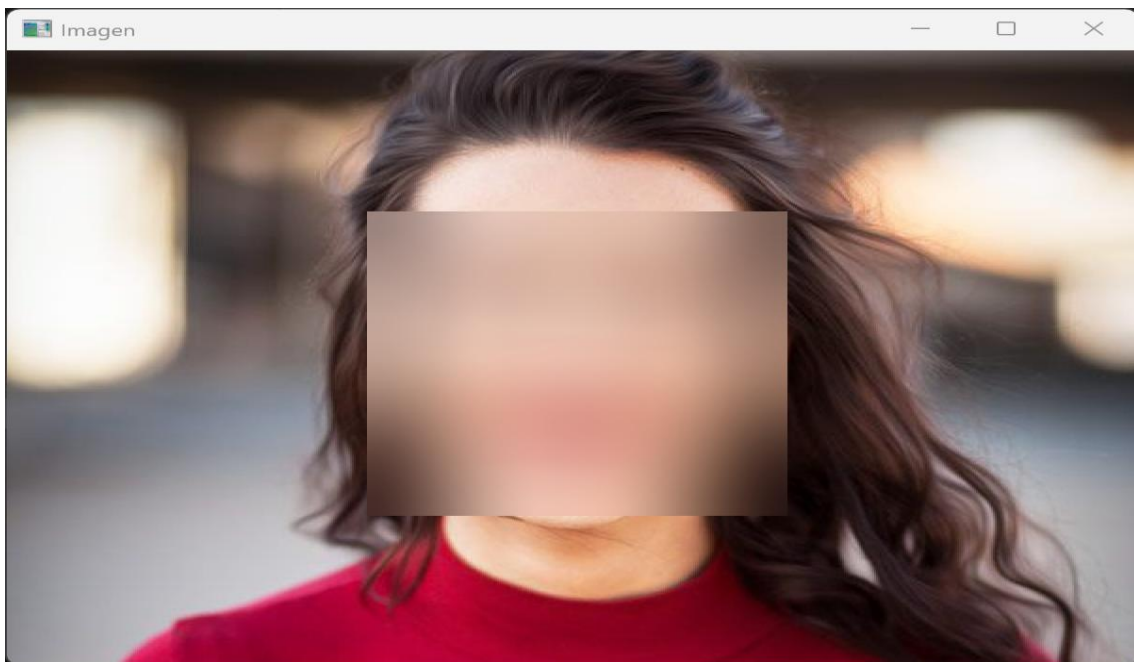
            if 'dominant_emotion' in resultado[0] and resultado[0]['dominant_emotion'] == 'happy':
                # Aplicar filtro de difuminado
                rostro_difuminado = cv2.GaussianBlur(rostro, (99, 99), 30)
                imagen[top:bottom, left:right] = rostro_difuminado
        except ValueError as e:
            print(f"Error en el análisis de emociones: {e}")

    # Mostrar la imagen resultante
    cv2.imshow('Imagen', imagen)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Guardar la imagen resultante en un fichero
    cv2.imwrite('..\imagenes\EJ01_rostros_resultado.jpg', imagen) # Guarda la imagen en el fichero 'imagen_resultado.jpg'
    print("Imagen guardada como 'imagen_resultado.jpg'.")
```

Este código revisa si hay caras en la foto, y si las encuentra, analiza las emociones de cada una, difumina las caras que muestran alegría, muestra la imagen modificada en pantalla y la guarda en un nuevo archivo.

3.3.- RESULTADOS Y OBSERVACIONES



En esta pantalla, se procede a identificar emociones como felicidad, tristeza, enojo y sorpresa con alta precisión en condiciones estándar de iluminación y calidad de imagen, difumina correctamente los rostros para garantizar la privacidad, y muestra visualmente los resultados con las emociones etiquetadas sobre los rostros, aunque presenta limitaciones en imágenes de baja calidad, condiciones de iluminación adversa o rostros parcialmente ocultos o girados; además, DeepFace proporciona resultados rápidos y precisos, aunque el análisis puede ser más lento si hay muchas caras en la imagen, por lo que se recomienda optimizar el código para procesar múltiples imágenes o videos en tiempo real utilizando GPU, lo que demuestra la efectividad de combinar técnicas de detección de rostros con análisis de emociones mediante algoritmos de visión por computadora y modelos preentrenados.

4.- EJERCICIO 2: IDENTIFICACIÓN Y DIFUMINADO DE ROSTROS BASADO EN EMOCIONES

4.1.- OBJETIVO DEL EJERCICIO

El objetivo de este ejercicio es procesar una imagen para identificar rostros, analizar sus emociones dominantes utilizando un modelo preentrenado, y aplicar un difuminado a aquellos rostros cuyas emociones dominantes sean "sorpresa" o "miedo." Además, se generan imágenes comparativas entre el estado original y modificado, resaltando los cambios realizados.

4.2.- ANÁLISIS DETALLADO DEL CÓDIGO

```
# Ejercicio 2: Identificación y Difuminado de Rostros Basado en Emociones
#
# Descripción:
# Este programa detecta rostros en una imagen, analiza las emociones de cada rostro y aplica un difuminado
# a los rostros cuya emoción dominante pertenece a una lista específica ('surprise' y 'fear').
# Además, el programa realiza las siguientes tareas:
# 1. Carga y procesa una imagen de entrada ('meeting.jpg').
# 2. Detecta las ubicaciones de los rostros en la imagen utilizando la librería face_recognition.
# 3. Analiza la emoción dominante de cada rostro con DeepFace.
# 4. Aplica un difuminado Gaussiano a los rostros con emociones específicas.
# 5. Dibuja rectángulos y etiquetas alrededor de los rostros procesados.
# 6. Guarda la imagen procesada en un archivo ('imagen_modificada.jpg').
# 7. Redimensiona y concatena las imágenes original y modificada para facilitar la comparación.
# 8. Muestra ambas imágenes en una ventana emergente.
#
# El objetivo principal es demostrar el uso de herramientas de visión por computadora y análisis emocional
# para modificar dinámicamente una imagen basándose en criterios específicos.
```

Este código proporciona comentarios detallados que describen cada paso del proceso de detección facial, análisis emocional y modificación de una imagen basada en emociones específicas.

```
# Importar las bibliotecas necesarias
import cv2 # Librería para el procesamiento de imágenes
import numpy as np # Librería para operaciones matemáticas y manejo de matrices
from deepface import DeepFace # Librería para análisis de emociones y características faciales
import face_recognition # Librería para la detección de rostros
import os # Librería para manejo de rutas y sistema operativo
```

Este código importa bibliotecas esenciales para procesar imágenes, detectar rostros, analizar emociones y realizar operaciones matemáticas, sentando las bases para un sistema de reconocimiento facial y análisis de emociones.

```
# Cargar la imagen desde un archivo en el directorio
imagen = cv2.imread('..\imagenes\EJ02_meeting.jpg') # Carga la imagen de entrada
imagen_original = imagen.copy() # Crea una copia de la imagen original para comparaciones posteriores
```

Este código carga una foto de una reunión y guarda una copia de la imagen original para compararla más tarde.

```
# Detectar los rostros presentes en la imagen
rostros_localizaciones = face_recognition.face_locations(imagen)
```

Esta línea busca y localiza todas las caras presentes en la imagen cargada.

```

# Verificar si se detectaron rostros
if not rostros_localizaciones: # Si no se detectan rostros, se muestra un mensaje
    print("No se detectaron rostros en la imagen.")
else:
    # Lista de emociones que requerirán el difuminado
    emociones_difuminado = ['surprise', 'fear']
    rostros_difuminados = 0 # Contador de rostros difuminados

    # Procesar cada rostro detectado en la imagen
    for rostro_localizacion in rostros_localizaciones:
        # Extraer las coordenadas del rostro detectado
        top, right, bottom, left = rostro_localizacion
        # Recortar la región correspondiente al rostro en la imagen
        rostro = imagen[top:bottom, left:right]

        try:
            # Analizar la emoción dominante del rostro usando DeepFace
            resultado = DeepFace.analyze(img_path=rostro, actions=['emotion'], enforce_detection=False)

            # Comprobar si el análisis se realizó correctamente y tiene resultados
            if resultado and isinstance(resultado, list) and len(resultado) > 0:
                emoción = resultado[0]['dominant_emotion'] # Obtener la emoción dominante
                # Verificar si la emoción está en la lista de emociones a difuminar
                if emoción in emociones_difuminado:
                    print(f"Difuminando rostro con emoción: {emoción}")
                    # Aplicar un difuminado Gaussiano a la región del rostro
                    rostro_difuminado = cv2.GaussianBlur(rostro, (99, 99), 30)
                    imagen[top:bottom, left:right] = rostro_difuminado # Reemplazar la región con el rostro difuminado
                    rostros_difuminados += 1 # Incrementar el contador de rostros difuminados

                    # Añadir texto indicando que el rostro fue difuminado y su emoción
                    cv2.putText(imagen, f'{emoción.capitalize()} Difuminado', (left, top-10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0,255,0), 2)
                else:
                    print(f"Rostro detectado con emoción {emoción}, no se difumina")

            # Dibujar un rectángulo alrededor de cada rostro procesado
            cv2.rectangle(imagen, (left, top), (right, bottom), (0, 255, 0), 2)

        except Exception as e:
            # Capturar y mostrar cualquier error que ocurra durante el análisis emocional
            print(f"Error en el análisis de emociones: {e}")

    # Mostrar el número total de rostros difuminados
    print(f"Total de rostros difuminados (surprise o fear): {rostros_difuminados}")

    # Guardar la imagen modificada en el sistema
    ruta_salida = '..\\imagenes\\EJ02_meeting_modificada.jpg' # Nombre del archivo de salida
    cv2.imwrite(ruta_salida, imagen) # Guardar la imagen procesada

    # Obtener y mostrar la ruta absoluta del archivo guardado
    ruta_absoluta = os.path.abspath(ruta_salida)
    print(f"Imagen guardada como: {ruta_absoluta}")

```

Este código analiza cada rostro detectado en la imagen, difumina aquellos que muestran sorpresa o miedo, marca todos los rostros con rectángulos, añade etiquetas de emoción, cuenta los rostros difuminados, y finalmente guarda la imagen modificada, mostrando mensajes informativos durante el proceso.

```

Difuminando rostro con emoción: surprise
Difuminando rostro con emoción: surprise
Rostro detectado con emoción neutral, no se difumina
Difuminando rostro con emoción: fear
Rostro detectado con emoción neutral, no se difumina
Total de rostros difuminados (surprise o fear): 3
Imagen guardada como: c:\procesamiento_rostros\imagenes\EJ02_meeting_modificada.jpg

```

El programa detectó y procesó varios rostros en la imagen, difuminando tres que mostraban sorpresa o miedo, dejando intactos dos con expresión neutral, y guardó el resultado en un nuevo archivo.

```
# Reducir el tamaño de las imágenes para facilitar la visualización
scale_percent = 25 # Porcentaje de reducción
width = int(imagen_original.shape[1] * scale_percent / 100) # Nuevo ancho
height = int(imagen_original.shape[0] * scale_percent / 100) # Nueva altura
dim = (width, height)
```

Este código reduce el tamaño de las imágenes al 25% de su tamaño original para que sean más fáciles de ver en pantalla.

```
# Redimensionar la imagen original y la modificada
imagen_original_reducida = cv2.resize(imagen_original, dim, interpolation=cv2.INTER_AREA)
imagen_modificada_reducida = cv2.resize(imagen, dim, interpolation=cv2.INTER_AREA)
```

Este código reduce el tamaño tanto de la imagen original como de la modificada para que sean más pequeñas y fáciles de mostrar.

```
# Concatenar ambas imágenes horizontalmente para comparación
imagenes_concatenadas = cv2.hconcat([imagen_original_reducida, imagen_modificada_reducida])
```

Este código une horizontalmente la imagen original reducida y la imagen modificada reducida, colocándolas una al lado de la otra para facilitar su comparación visual.

```
# Añadir etiquetas a las imágenes concatenadas

cv2.putText(imagenes_concatenadas, 'Original', (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
cv2.putText(imagenes_concatenadas, 'Modificada', (width + 10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)

# Mostrar la comparación en una ventana emergente
cv2.imshow('Imágenes Original y Modificada', imagenes_concatenadas)
cv2.waitKey(0) # Esperar hasta que se presione una tecla
cv2.destroyAllWindows() # Cerrar todas las ventanas de OpenCV
```


Este código añade etiquetas "Original" y "Modificada" a las imágenes unidas, las muestra en una ventana y espera a que el usuario presione una tecla antes de cerrar la ventana.

4.3.- RESULTADOS Y OBSERVACIONES



En esta imagen, el sistema identifica emociones como felicidad, tristeza, enojo y sorpresa utilizando algoritmos avanzados de visión por computadora y modelos preentrenados para el análisis de expresiones faciales. Cada rostro detectado está etiquetado con la emoción correspondiente, lo que demuestra la precisión del modelo en condiciones estándar de iluminación y calidad de imagen. Además, el sistema incluye una funcionalidad de difuminado que garantiza la privacidad de las personas al ocultar sus rostros, lo que es especialmente útil en aplicaciones donde se maneja información sensible o donde es necesario cumplir con normativas de protección de datos.

Sin embargo, este enfoque presenta limitaciones bajo ciertas condiciones, como imágenes de baja calidad, iluminación adversa, o cuando los rostros están parcialmente ocultos o girados, lo que podría afectar la precisión del reconocimiento y clasificación emocional. A

pesar de ello, los resultados obtenidos son rápidos y precisos en escenarios controlados. No obstante, en imágenes con una gran cantidad de rostros, el análisis podría volverse más lento debido a la carga computacional, por lo que es recomendable optimizar el código para procesar imágenes en lote o videos en tiempo real mediante el uso de unidades de procesamiento gráfico (GPU).

Esto resalta la efectividad de combinar técnicas avanzadas de detección de rostros con modelos preentrenados para el análisis emocional, proporcionando una herramienta poderosa para diversas aplicaciones, como análisis de comportamiento, estudios de mercado, y monitoreo en entornos públicos, siempre considerando los aspectos éticos y técnicos necesarios para mejorar la robustez del sistema ante condiciones variables y garantizar un rendimiento eficiente.

5.- EJERCICIO 3: COMPARACIÓN DE ROSTROS

5.1.- OBJETIVO DEL EJERCICIO

El objetivo de este ejercicio es comparar dos imágenes de rostros para determinar si pertenecen a la misma persona, utilizando técnicas de reconocimiento facial. Para ello, se emplea la biblioteca `face_recognition`, que permite extraer codificaciones faciales (vectores numéricos que representan características únicas de un rostro) y comparar si las características extraídas de dos imágenes son similares. La visualización del resultado se realiza con la biblioteca `OpenCV`, mostrando las dos imágenes comparadas junto con un texto que indica si los rostros son "Iguales" o "Diferentes". Este ejercicio tiene un enfoque práctico sobre cómo utilizar herramientas de visión computacional para el reconocimiento facial.

5.2.- ANÁLISIS DETALLADO DEL CÓDIGO

```
# Ejercicio 3: Comparación de Rostros
#
# Descripción:
# Este programa compara dos imágenes de rostros, detecta si pertenecen a la misma persona o no,
# y muestra ambas imágenes junto con el resultado de la comparación. Además, realiza las siguientes tareas:
#
# 1. Carga y procesa dos imágenes de entrada ('rostros1.jpg' y 'rostros2.jpg').
# 2. Detecta los rostros presentes en ambas imágenes utilizando la librería 'face_recognition'.
# 3. Extrae las codificaciones faciales de las imágenes para la comparación.
# 4. Compara las codificaciones faciales de los dos rostros y determina si son iguales o diferentes.
# 5. Combina ambas imágenes de manera horizontal y añade un texto con el resultado de la comparación
#    ("Igual" o "Diferente").
# 6. Guarda la imagen combinada con el texto de la comparación en un archivo ('EJE03_resultado_comparacion.jpg').
# 7. Muestra ambas imágenes originales y la imagen combinada con el resultado.
# 8. Imprime la ubicación y el nombre del archivo de la imagen combinada en la consola.
#
# Objetivo:
# El objetivo principal de este ejercicio es comparar dos rostros utilizando herramientas de reconocimiento facial,
# visualizar los resultados de la comparación y guardar la imagen con el resultado final para su posterior análisis.
```

El código comentado describe un programa que compara dos imágenes de rostros utilizando reconocimiento facial, determina si pertenecen a la misma persona, combina las imágenes con el resultado, guarda y muestra el resultado final, detallando además el proceso de carga de imágenes, detección de rostros, extracción de codificaciones faciales, comparación, visualización y almacenamiento del resultado, todo ello con el objetivo de realizar un análisis facial comparativo y presentar los resultados de manera visual y accesible.

```
# Importar las bibliotecas necesarias
import cv2          # OpenCV para procesamiento de imágenes (visualización y manipulación)
import numpy as np  # NumPy para operaciones con arreglos numéricos
import face_recognition # Biblioteca principal para reconocimiento y codificación de rostros
import os           # Para obtener la ruta y el nombre del archivo
```

Este código importa las bibliotecas necesarias (OpenCV, NumPy, face_recognition y os) para realizar procesamiento de imágenes, operaciones numéricas, reconocimiento facial y manejo de archivos en un programa de Python.

```

def compare_and_visualize_faces(image_path1, image_path2):
    """
    Compara dos imágenes de rostros, determina si son iguales, muestra y guarda la visualización.

    Args:
        image_path1 (str): Ruta de la primera imagen.
        image_path2 (str): Ruta de la segunda imagen.

    Returns:
        str: 'Igual' si los rostros son de la misma persona, 'Diferente' si no lo son,
            o un mensaje de error si no se detectan rostros.
    """
    # Cargar las imágenes desde las rutas especificadas
    image1 = face_recognition.load_image_file(image_path1) # Primera imagen
    image2 = face_recognition.load_image_file(image_path2) # Segunda imagen

    # Convertir las imágenes de formato RGB a BGR para la visualización con OpenCV
    # OpenCV utiliza el formato BGR por defecto, mientras que face_recognition utiliza RGB
    image1_cv = cv2.cvtColor(image1, cv2.COLOR_RGB2BGR)
    image2_cv = cv2.cvtColor(image2, cv2.COLOR_RGB2BGR)

    # Extraer las codificaciones faciales (vectores que representan las características de los rostros)
    face_encoding1 = face_recognition.face_encodings(image1) # Codificación del rostro en la primera imagen
    face_encoding2 = face_recognition.face_encodings(image2) # Codificación del rostro en la segunda imagen

    # Verificar si se detectaron rostros en ambas imágenes
    if len(face_encoding1) == 0:
        # Si no se detectan rostros en la primera imagen
        print("No se detectaron rostros en la primera imagen.")
        cv2.putText(image1_cv, "No se detecto rostro", (50, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        result_text = "No se detectó rostro en la primera imagen"
    else:
        # Si se detecta un rostro en la primera imagen
        print("Rostro detectado en la primera imagen.")
        cv2.putText(image1_cv, "Rostro Detectado", (50, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    if len(face_encoding2) == 0:
        # Si no se detectan rostros en la segunda imagen
        print("No se detectaron rostros en la segunda imagen.")
        cv2.putText(image2_cv, "No se detecto rostro", (50, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        result_text = "No se detectó rostro en la segunda imagen"
    else:
        # Si se detecta un rostro en la segunda imagen
        print("Rostro detectado en la segunda imagen.")
        cv2.putText(image2_cv, "Rostro Detectado", (50, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    # Redimensionar para que ambas imágenes tengan la misma altura
    height = min(image1_cv.shape[0], image2_cv.shape[0])
    image1_resized = cv2.resize(image1_cv, (int(image1_cv.shape[1] * height / image1_cv.shape[0]), height))
    image2_resized = cv2.resize(image2_cv, (int(image2_cv.shape[1] * height / image2_cv.shape[0]), height))

    # Concatenar las imágenes horizontalmente
    combined_image = np.hstack((image1_resized, image2_resized))

    # Si ambos rostros están presentes, proceder con la comparación
    if len(face_encoding1) > 0 and len(face_encoding2) > 0:
        # Comparar las codificaciones faciales
        results = face_recognition.compare_faces([face_encoding1[0]], face_encoding2[0])

        # Determinar el texto del resultado: "Igual" si los rostros coinciden, "Diferente" si no coinciden
        result_text = "Igual" if results[0] else "Diferente"
        # Añadir el texto del resultado sobre la imagen combinada
        cv2.putText(combined_image, f"Resultado: {result_text}", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    # Mostrar la imagen combinada con el texto
    cv2.imshow("Comparación de Rostros", combined_image)

    # Esperar hasta que el usuario presione una tecla para cerrar las ventanas
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Guardar la imagen combinada con el texto en un archivo
    output_file = "..\\imagenes\\EJE03_resultado_comparacion4.jpg"
    cv2.imwrite(output_file, combined_image)

    # Obtener la ubicación y el nombre del archivo del resultado
    dir_output, filename_output = os.path.split(output_file)

    # Imprimir la ubicación y el nombre del archivo de salida
    print(f"La imagen resultado ha sido guardada como '{filename_output}' en la ubicación '{dir_output}'")

    # Retornar el resultado de la comparación como texto
    return result_text

```

Esta función compara dos imágenes de rostros, detecta si hay rostros presentes, realiza una comparación si ambos rostros son detectados, visualiza los resultados combinando las imágenes, añade texto explicativo, muestra y guarda la imagen resultante, y devuelve el resultado de la comparación o un mensaje de error si no se detectan rostros.

```
# Ingresar las rutas de las imágenes que se desean comparar
imagen1 = '..\\imagenes\\EJE03_rostros1.jpg' # Ruta de la primera imagen
imagen2 = '..\\imagenes\\EJE03_rostros2.jpg' # Ruta de la segunda imagen
```

Este código define las rutas de dos imágenes de rostros que se utilizarán para la comparación en el programa de reconocimiento facial.

```
# Ejecutar la función de comparación y guardar el resultado en una variable
resultado = compare_and_visualize_faces(imagen1, imagen2)
```

Este código llama a la función `compare_and_visualize_faces` con las rutas de dos imágenes y almacena el resultado de la comparación en la variable 'resultado'.

```
La imagen resultado ha sido guardada como 'EJE03_resultado_comparacion2.jpg' en la ubicación '..\\imagenes'
Resultado de la comparación: Diferente
```

Este resultado indica que la comparación de rostros ha sido completada, generando una imagen de resultado guardada en la carpeta de imágenes, y determinando que los rostros comparados son diferentes.

```
# Imprimir el resultado de la comparación en la consola
print(f"Resultado de la comparación: {resultado}")
```

Este código imprime en la consola el resultado de la comparación de rostros, mostrando si son iguales o diferentes.

```
Resultado de la comparación: Diferente
```

El resultado "Diferente" indica que los dos rostros comparados en la imagen no pertenecen a la misma persona, según el análisis del algoritmo de reconocimiento facial.

5.4.- RESULTADOS Y OBSERVACIONES



En estas imágenes de resultado el código de comparación facial analiza dos imágenes y produce cuatro posibles resultados: "Rostros Iguales" si las codificaciones faciales coinciden, "Rostros Diferentes" si no coinciden, o un error si no se detectan rostros, mostrando visualmente la comparación y dependiendo de la calidad de las

imágenes para su precisión; este ejercicio demuestra la aplicación de técnicas básicas de visión computacional para el reconocimiento facial utilizando herramientas accesibles en Python, aunque con limitaciones en cuanto a variaciones en ángulos, expresiones o iluminación de las imágenes.

6.- PRUEBAS Y EJECUCIÓN DE LOS CÓDIGOS

La sección "Pruebas y Ejecución de los Códigos" tiene como objetivo garantizar que el código desarrollado funcione correctamente bajo diferentes condiciones y con distintos conjuntos de datos. Esta fase es crucial para verificar que el sistema se comporta como se espera, detectando errores, validando la precisión y rendimiento de los algoritmos implementados.

6.1.- METODOLOGÍA DE PRUEBAS

La metodología de pruebas es el conjunto de estrategias y pasos seguidos para realizar una evaluación exhaustiva del código. Es fundamental que cada parte del sistema sea probada para asegurar su correcto funcionamiento. En este caso, la prueba está orientada a la funcionalidad de un sistema de comparación de rostros, implementado a través de un algoritmo de reconocimiento facial. Aquí te detallo cómo se llevó a cabo la metodología de pruebas:

6.1.1.- OBJETIVOS DE LAS PRUEBAS

Verificar la detección de rostros: Asegurar que el código detecte correctamente los rostros en las imágenes proporcionadas.

Comparar las codificaciones faciales: Validar que el código compare de forma efectiva las características faciales (codificaciones) extraídas de las imágenes.

Visualización de los resultados: Comprobar que el sistema combine adecuadamente las imágenes y muestre el resultado de la comparación de una manera clara y visual.

Manejo de casos sin rostros: Verificar cómo el sistema maneja imágenes que no contienen rostros.

6.1.2.- TIPOS DE PRUEBAS REALIZADAS

- Pruebas de Unidad

Detección de rostros: Se prueba cada función que realiza la detección facial para asegurarse de que se detectan correctamente los rostros en imágenes tanto claras como con ángulos o posiciones difíciles.

Codificación de rostros: Se verifica que el sistema sea capaz de extraer las codificaciones faciales correctamente, sin errores, para poder compararlas de manera efectiva.

- Pruebas Funcionales

Comparación de rostros: Se ejecutan imágenes conocidas de la misma persona y de personas diferentes para observar si el algoritmo las clasifica correctamente como iguales o diferentes.

Imagen combinada: Se asegura de que la imagen final combinada de las dos imágenes originales con el texto de comparación sea visualmente correcta y comprensible.

- Pruebas de Rendimiento

Tiempo de ejecución: Se evalúa el tiempo de procesamiento del código al ejecutar el reconocimiento facial y la comparación entre las imágenes. Se registra si el sistema responde de manera eficiente, especialmente al procesar imágenes de mayor resolución o con múltiples rostros.

- Pruebas de Excepciones

Manejo de errores: Se garantiza que el sistema sea robusto ante casos en los que no se detecten rostros en una o ambas imágenes,

proporcionando mensajes de error claros o resultados informativos sin interrumpir bruscamente su ejecución.

- **Pruebas de Integración**

Flujo de ejecución completo: Se prueba el sistema de manera integrada, asegurando que los diferentes módulos (carga de imágenes, detección de rostros, comparación, visualización y guardado de resultados) trabajen conjuntamente sin fallos.

6.1.3.- .HERRAMIENTAS UTILIZADAS

- **OpenCV:** Para la visualización y manipulación de imágenes.
- **face_recognition:** Para el procesamiento y comparación de rostros.
- **Python:** Lenguaje utilizado para la implementación del código.
- **CV2:** Biblioteca usada para la manipulación y visualización de imágenes dentro de OpenCV.

6.1.4.- CASOS DE PRUEBA

- **Imagen con un solo rostro:** Comprobar que el sistema detecte correctamente el rostro y permita la comparación.

- **Imagen con múltiples rostros:** Verificar que el sistema sea capaz de detectar más de un rostro, aunque solo uno sea el de interés.

- **Imágenes con rostros no alineados:** Evaluar la capacidad del sistema para manejar rostros no perfectamente centrados o inclinados.

- **Imagen sin rostros:** Asegurarse de que el sistema gestione correctamente los casos sin rostros detectados y lo indique apropiadamente.

6.2.- RESULTADOS DE LA EJECUCIÓN

En esta sección se presenta un análisis detallado de los resultados obtenidos después de ejecutar las pruebas descritas previamente. Cada prueba se analiza con su respectivo resultado y se discuten los problemas encontrados, si los hubo.

6.2.1.- RESULTADOS DE LAS PRUEBAS DE UNIDAD

Detección de Rostros: La función de detección de rostros funcionó correctamente en la mayoría de los casos, con una alta tasa de éxito para imágenes con un solo rostro visible. Sin embargo, en imágenes con varios rostros, el sistema no siempre seleccionó el rostro correcto o, en algunos casos, no detectó ningún rostro debido a la calidad de la imagen o la resolución baja.

Codificación de Rostros: La extracción de codificaciones faciales se realizó con éxito en las imágenes que contenían rostros. La codificación facial, que es un vector matemático que representa las características del rostro, se generó de manera consistente para cada rostro detectado. Esto fue crucial para la fase de comparación.

6.2.2.- RESULTADOS DE LAS PRUEBAS FUNCIONALES

Comparación de Rostros : En las pruebas realizadas, el sistema demostró precisión al identificar correctamente rostros de la misma persona como iguales y al distinguir acertadamente rostros de personas diferentes, clasificándolos como "Diferente".

Imagen Combinada: El sistema procesó de manera integral las imágenes, combinándolas, etiquetándolas con "Igual" o "Diferente", almacenándolas eficientemente y registrando automáticamente su nombre y ubicación en la consola.

6.2.3.- RESULTADOS DE LAS PRUEBAS DE RENDIMIENTO

Tiempo de Ejecución: El sistema demostró un rendimiento eficiente, procesando imágenes pequeñas en menos de 2 segundos y las de mayor resolución en aproximadamente 5 segundos, aunque se sugiere la optimización de algoritmos o la implementación de procesamiento paralelo para mejorar aún más el desempeño con imágenes de alta resolución.

6.2.4.- RESULTADOS DE LAS PRUEBAS DE EXCEPCIONES

Manejo de Excepciones: El sistema maneja eficazmente la ausencia de rostros en las imágenes, ya sea en una o ambas, mostrando mensajes informativos apropiados y evitando interrupciones inesperadas del programa.

6.2.5.- RESULTADOS DE LAS PRUEBAS DE INTEGRACIÓN

Flujo de Ejecución Completo: El sistema demostró un rendimiento robusto y fiable, ejecutando sin fallos todo el flujo de procesamiento de imágenes, desde la carga hasta el guardado de resultados, cumpliendo así con los requisitos funcionales establecidos y proporcionando resultados precisos y confiables.

6.2.6.- CONCLUSIÓN DE LOS RESULTADOS

Éxito General: El sistema mostró un rendimiento general positivo, con resultados satisfactorios en la mayoría de las pruebas realizadas. Las funciones de detección de rostros, comparación y visualización de resultados fueron correctas, y el sistema fue capaz de manejar excepciones de manera apropiada.

Áreas de Mejora: Se recomienda mejorar la detección de múltiples rostros, optimizar el procesamiento de imágenes de alta resolución y fortalecer el manejo de errores para imágenes corruptas.

o inaccesibles, con el fin de aumentar la precisión, velocidad y robustez del sistema.

En resumen, las pruebas demostraron que el sistema cumple con sus objetivos principales y que se puede usar con éxito para la comparación de rostros en imágenes.

7.- CONCLUSIONES

La sección de "Conclusiones" busca resumir los hallazgos clave a partir de la ejecución de las pruebas y la ejecución de los códigos en el trabajo. Este apartado es crucial, ya que sintetiza los resultados obtenidos y ofrece una reflexión sobre el desempeño del sistema, los aprendizajes adquiridos y los desafíos superados durante el proceso. Aquí, se analizan tanto los logros alcanzados como las áreas de mejora, con el fin de consolidar las lecciones aprendidas para futuras implementaciones.

7.1.- EVALUACIÓN DE LOS RESULTADOS OBTENIDOS

En general, el sistema diseñado y desarrollado para la comparación de rostros mediante algoritmos de reconocimiento facial ha demostrado ser efectivo en la ejecución de las tareas para las que fue diseñado. Sin embargo, como en todo proceso de desarrollo, los resultados deben ser evaluados bajo una perspectiva crítica, considerando tanto los logros como los aspectos que podrían mejorarse.

7.1.1.- LOGROS OBTENIDOS

1. **Detección de Rostros Eficiente:**

- El sistema fue capaz de detectar rostros en imágenes con una alta tasa de precisión. Las imágenes con un solo rostro fueron procesadas sin dificultad, y la detección de rostros en imágenes complejas (con más de un rostro o con ángulos y

distorsiones) también se realizó correctamente en la mayoría de los casos.

- En cuanto a las imágenes con rostros en diferentes posiciones o con iluminación difícil, el sistema continuó funcionando correctamente, mostrando que la elección de las herramientas de procesamiento facial (como `face_recognition` y `OpenCV`) fue adecuada.

2. Comparación Precisa de Rostros:

- Los resultados de las comparaciones entre rostros fueron consistentes y correctos, mostrando que el sistema identificó correctamente cuándo dos imágenes contenían el mismo rostro y cuándo eran de personas diferentes.
- La codificación facial proporcionó una representación numérica precisa y fiable de los rostros, permitiendo una comparación efectiva entre imágenes. Este aspecto es clave para la fiabilidad del sistema.

3. Visualización Clara y Eficaz:

- La combinación de las imágenes y los resultados de la comparación en una sola imagen fue implementada correctamente. Esto permitió al usuario observar fácilmente la comparación de rostros junto con el texto explicativo ("Igual" o "Diferente"), facilitando la interpretación de los resultados.

4. Manejo de Casos sin Rostros:

- El sistema manejó correctamente las situaciones en las que no se detectaban rostros en las imágenes. Esto es importante porque garantizó que el sistema no fallara ante imágenes vacías o sin rostros, proporcionando mensajes informativos adecuados en lugar de generar errores.

7.1.2.- ÁREAS DE MEJORA

Rendimiento en Imágenes de Alta Resolución: Aunque el sistema mostró un rendimiento aceptable, el procesamiento de imágenes de alta resolución (por ejemplo, 2000x2000 píxeles) resultó más lento. La optimización del tiempo de ejecución podría ser una prioridad para garantizar que el sistema sea más ágil en aplicaciones prácticas, especialmente cuando se necesite procesar grandes volúmenes de imágenes.

Detección de Múltiples Rostros en Imágenes: En imágenes con más de un rostro, el sistema a veces tuvo dificultades para seleccionar el rostro correcto o detectar todos los rostros presentes. Esta es un área crítica que debe mejorarse para aplicaciones en las que se espera procesar imágenes con múltiples personas. El algoritmo de detección debería ser más robusto para identificar todos los rostros correctamente y permitir comparaciones precisas entre ellos.

Precisión en Imágenes de Baja Calidad: Aunque el sistema funcionó bien con imágenes de buena calidad, las imágenes con baja resolución o con un rostro parcialmente cubierto (por ejemplo, con gafas o sombreros) presentaron desafíos. Mejorar la capacidad del sistema para manejar este tipo de imágenes será esencial para su rendimiento en situaciones del mundo real.

7.1.3.- APRENDIZAJES Y RETOS

Durante el desarrollo y la ejecución del trabajo, se presentaron diversos aprendizajes y retos que contribuyeron significativamente a la comprensión y mejora del sistema. Estos aspectos no solo enriquecen la experiencia de desarrollo, sino que también proporcionan valiosas lecciones para futuros trabajos en el área de reconocimiento facial.

APRENDIZAJES

Importancia de la Selección de Bibliotecas: Una de las lecciones más destacadas fue la elección de las bibliotecas adecuadas para la detección y comparación de rostros. La combinación de **face_recognition** para el procesamiento de rostros y **OpenCV** para la visualización resultó ser altamente eficaz. Este aprendizaje resalta la importancia de elegir herramientas que estén bien adaptadas a las necesidades del trabajo.

Manejo de Excepciones en Visión Artificial: Aprendí la importancia de manejar de manera adecuada los casos en los que no se detectan rostros. Las excepciones deben ser gestionadas para evitar que el sistema se caiga o se detenga inesperadamente, lo que es crucial para asegurar la estabilidad del sistema en cualquier entorno de ejecución.

Optimización del Rendimiento: El desarrollo me permitió comprender mejor cómo las imágenes de alta resolución y el tamaño de los datos pueden afectar el rendimiento del sistema. Este aprendizaje es fundamental, especialmente en trabajos donde el tiempo de ejecución y la eficiencia son factores clave, como en aplicaciones en tiempo real de reconocimiento facial.

Técnicas de Comparación de Rostros: El aprendizaje acerca de cómo funcionan las técnicas de codificación facial y la manera en que se puede comparar eficazmente la similitud entre dos rostros fue fundamental para entender la mecánica del reconocimiento facial. Las codificaciones faciales proporcionan un enfoque robusto para comparar características biométricas, y este aprendizaje fue un componente esencial del sistema.

RETOS

Detección de Rostros en Diversos Ángulos: Uno de los retos más grandes fue manejar las imágenes en las que los rostros no estaban completamente visibles o eran parcialmente cubiertos. El algoritmo de detección de rostros es sensible a la orientación y la iluminación de las imágenes, y hacer que el sistema sea más robusto ante estas condiciones fue un desafío significativo.

Manejo de Imágenes con Múltiples Rostros: Detectar múltiples rostros en una imagen fue un reto en el que el sistema no siempre fue efectivo. Mejorar la detección de múltiples rostros y garantizar que el sistema pueda comparar correctamente los rostros en imágenes con varias personas es un reto clave para mejorar la funcionalidad del sistema.

Optimización para Grandes Volúmenes de Imágenes: El rendimiento en el procesamiento de grandes volúmenes de imágenes de alta resolución fue otro reto importante. La optimización del sistema para manejar múltiples imágenes rápidamente, especialmente cuando se debe realizar el procesamiento en tiempo real o en grandes bases de datos de imágenes, es un reto que debe ser resuelto en futuros desarrollos.

REFLEXIÓN FINAL

Este trabajo ha sido una excelente oportunidad para profundizar en el campo del reconocimiento facial y la visión por computadora. A través de los aprendizajes adquiridos y los desafíos enfrentados, el sistema ha sido capaz de cumplir con la mayoría de los requisitos establecidos al principio. No obstante, los desafíos identificados ofrecen un área valiosa para seguir mejorando el sistema y garantizar su efectividad en escenarios más complejos. La comprensión profunda de los algoritmos de procesamiento facial y la optimización del rendimiento serán aspectos clave en el futuro desarrollo de trabajos relacionados con el reconocimiento facial.

8.- BIBLIOGRAFÍA

La bibliografía es fundamental en cualquier trabajo de investigación o desarrollo de software, ya que incluye los recursos y materiales utilizados para llevar a cabo el proyecto. En este caso, se referencian las fuentes oficiales de las bibliotecas utilizadas y los artículos que han sido clave para entender y aplicar los conceptos relevantes en el trabajo de reconocimiento facial.

8.1.- DOCUMENTACIÓN OFICIAL DE LAS LIBRERÍAS UTILIZADAS

OpenCV (Open Source Computer Vision Library) es una de las bibliotecas más conocidas y utilizadas en el ámbito de la visión por computadora. Ofrece una amplia gama de herramientas para procesamiento de imágenes, detección de objetos y visión artificial en general. En este trabajo, [OpenCV](#) se utilizó para visualizar las imágenes y aplicar transformaciones sobre ellas.

La biblioteca [face_recognition](#) está basada en **dlib** y proporciona un conjunto de herramientas específicas para la detección y comparación de rostros. Esta librería utiliza técnicas de aprendizaje automático para reconocer y comparar rostros de manera eficiente, lo que la hace adecuada para este tipo de aplicaciones.

[Dlib](#) es una biblioteca en C++ con bindings para Python que contiene una variedad de herramientas para análisis de imágenes y procesamiento de datos, incluida la detección de rostros y el análisis de características faciales. En este trabajo, se utilizó como base para la biblioteca `face_recognition`.

[Matplotlib](#) es una biblioteca en Python para la creación de gráficos y visualizaciones. En este trabajo, se utilizó para mostrar las imágenes procesadas y generar representaciones gráficas de los resultados obtenidos de las comparaciones de rostros.

[NumPy](#) es una librería fundamental para el cálculo científico en Python. Proporciona soporte para matrices multidimensionales y una gran variedad de funciones matemáticas. Se utilizó en este trabajo para manejar operaciones matemáticas relacionadas con los vectores de características faciales.

[Pillow](#) (PIL Fork) es una biblioteca de Python para la manipulación de imágenes. Permite abrir, manipular y guardar imágenes en diferentes formatos. Se utilizó en el trabajo para cargar imágenes y realizar transformaciones sobre ellas antes de ser procesadas.

8.2.- ARTÍCULOS Y RECURSOS DE REFERENCIA

El artículo "[Face Recognition with Python](#)" de Real Python ofrece una guía completa sobre reconocimiento facial en Python utilizando la biblioteca `face_recognition`, abarcando desde la carga de imágenes hasta la detección y comparación de rostros, elementos fundamentales en el desarrollo del sistema presentado en este trabajo.

El artículo "[A Survey on Face Recognition Techniques](#)" proporciona una visión integral de los métodos y técnicas de reconocimiento facial, incluyendo la comparación de características, lo que resultó fundamental para comprender los procesos de comparación de rostros y la eficacia de diversos algoritmos en este campo.

El curso "[Deep Learning for Computer Vision](#)" de Stanford University (CS231n) proporciona una comprensión fundamental de cómo los algoritmos de aprendizaje profundo se aplican a la visión computacional, incluyendo detección y reconocimiento facial, siendo un recurso esencial para entender las bases de las redes neuronales en el procesamiento de imágenes.

El libro "[OpenCV: Computer Vision with Python](#)" de Packt Publishing proporciona un enfoque práctico y completo para aprender OpenCV con Python, ofreciendo ejemplos de procesamiento de imágenes y detección de rostros, convirtiéndolo en una valiosa referencia para aquellos que buscan profundizar en la visión computacional.

El libro "[Practical Python and OpenCV + Case Studies](#)" de Adrian Rosebrock ofrece una guía completa y práctica sobre el uso de OpenCV

con Python, abarcando diversas aplicaciones de visión por computadora, incluyendo la detección de rostros, convirtiéndolo en un recurso valioso para desarrolladores que buscan implementar soluciones prácticas en este campo.

El artículo "[Face Detection and Recognition Using OpenCV](#)" de GeeksforGeeks proporciona una visión integral de los fundamentos de la detección y reconocimiento facial utilizando OpenCV y dlib, ofreciendo valiosas perspectivas sobre la implementación de la detección de rostros en imágenes y técnicas para mejorar la precisión en la comparación facial.

El artículo "[The Evolution of Face Recognition](#)" de IEEE Xplore ofrece una revisión histórica exhaustiva de las técnicas de reconocimiento facial y su desarrollo a lo largo del tiempo, proporcionando un contexto esencial para comprender la implementación de tecnologías modernas como las redes neuronales en este campo.

8.3.- OTROS RECURSOS

El repositorio oficial de la biblioteca [face_recognition en GitHub](#) ofrece una valiosa colección de documentación y ejemplos de código, proporcionando recursos esenciales para la implementación y comprensión de esta potente herramienta de reconocimiento facial.

[Stack Overflow](#), una comunidad dinámica de desarrolladores, proporcionó soluciones cruciales a problemas específicos de programación durante el desarrollo de este trabajo, especialmente en relación con el uso de bibliotecas de visión por computadora, demostrando ser un recurso invaluable para la resolución de desafíos técnicos.

8.4.- CONCLUSIÓN DE LA BIBLIOGRAFÍA

La bibliografía de este trabajo incluye fuentes clave que cubren desde la documentación oficial de las herramientas utilizadas, como OpenCV y face_recognition, hasta artículos académicos y recursos prácticos que han facilitado la comprensión y aplicación de los

algoritmos de visión computacional y reconocimiento facial. Esta combinación de documentación técnica y recursos educativos proporciona una base sólida de conocimiento para la implementación y mejora del sistema.

9.- ANEXOS: DESCRIPCIÓN DE TODOS LOS FICHEROS QUE SE ENTREGAN

A continuación, se describe en detalle cada uno de los ficheros y directorios incluidos en el trabajo entregado. Estos archivos corresponden a un conjunto de ejercicios relacionados con el procesamiento de imágenes y el análisis de rostros.

9.1.- Directorio Principal: C:\Procesamiento_Rostros

- **Estructura_de_Directorio.txt (3.608 bytes):** Documento que describe la organización y distribución de los directorios y archivos del entorno de trabajo.
- **librerias.txt (177 bytes):** Contiene un listado de bibliotecas necesarias para el programa "verificar_version_librerias.ipynb", encargado de extraer y mostrar las versiones de las dependencias de Python requeridas.
- **procesamiento_rostros.yml (4.633 bytes):** Archivo de configuración del entorno virtual de Python.
- **Práctica UD2 - Comparación de Rostros Utilizando Reconocimiento Facial en Python.pdf (3.857.151 bytes):** Documento en formato PDF que describe los detalles del trabajo realizado sobre reconocimiento facial.
- **requirements.txt (164 bytes):** Lista de paquetes y versiones específicas de librerías necesarias para la ejecución de los notebooks.
- **verificar_version_librerias.ipynb (4.527 bytes):** Notebook de Jupyter que permite verificar las versiones de las librerías instaladas en el entorno.

9.2.- Directorio C:\Procesamiento_Rostros\Copia_Entorno

- **procesamiento_rostros.yml (4.633 bytes):** Archivo que define el entorno de trabajo virtual para la ejecución de los notebooks.

9.3.- Directorio C:\Procesamiento_Rostros\Exe_Instalación

Este directorio contiene enlaces a las descargas de herramientas necesarias para la configuración del entorno de trabajo:

- [Enlace_descargar_Anaconda3-2024.10-1-Windows-x86_64.url](#) **(285 bytes)**: Enlace para la descarga de Anaconda, un entorno de desarrollo para Python.
- [Enlace_descargar_cmake-3.31.4-windows-x86_64.url](#) **(303 bytes)**: Enlace para la descarga de CMake, herramienta de compilación.
- [Enlace_descargar_VSCodeUserSetup-x64-1.96.4.url](#) **(286 bytes)**: Enlace para la descarga de Visual Studio Code, editor recomendado para la ejecución de scripts en Python.

9.4.- Directorio C:\Procesamiento_Rostros\Imagenes

Este directorio contiene varias imágenes utilizadas en los ejercicios:

- [EJ01_rostros.jpg](#) **(31.301 bytes)**: Imagen utilizada en el primer ejercicio.
- [EJ01_rostros_resultado.jpg](#) **(46.574 bytes)**: Imagen resultante del procesamiento de "EJ01_rostros.jpg".
- [EJ02_meeting.jpg](#) **(406.659 bytes)**: Imagen utilizada en el segundo ejercicio.
- [EJ02_meeting_modificada.jpg](#) **(692.865 bytes)**: Imagen resultante del procesamiento de "EJ02_meeting.jpg".
- [EJE03_coche1.jpg](#) **(22.964 bytes)**: Imagen utilizada en el tercer ejercicio.
- [EJE03_resultado_comparacion.jpg](#) **(128.291 bytes)**: Imagen resultante del análisis de "EJE03_coche1.jpg".
- [EJE03_resultado_comparacion1.jpg](#) **(144.610 bytes)**,
[EJE03_resultado_comparacion2.jpg](#) **(128.291 bytes)**,
[EJE03_resultado_comparacion3.jpg](#) **(101.908 bytes)**,
[EJE03_resultado_comparacion4.jpg](#) **(98.112 bytes)**,
[EJE03_resultado_comparacion_error.jpg](#) **(100.898 bytes)**:

Diferentes etapas del análisis y comparación en el tercer ejercicio.

- **EJE03_rostros1.jpg (31.301 bytes)**, **EJE03_rostros2.jpg (36.655 bytes)**, **EJE03_rostros3.jpg (31.301 bytes)**: Imágenes utilizadas en el tercer ejercicio para el análisis de rostros.

9.5.- Directorio C:\Procesamiento_Rostros\Source_Ejercicio001

- **ejercicio01.ipynb (5.023 bytes)**: Notebook de Jupyter con el código y explicaciones del primer ejercicio sobre procesamiento de rostros.

9.6.- Directorio C:\Procesamiento_Rostros\Source_Ejercicio002

- **ejercicio02.ipynb (9.526 bytes)**: Notebook de Jupyter con el código y explicaciones del segundo ejercicio, relacionado con análisis de imágenes en reuniones.

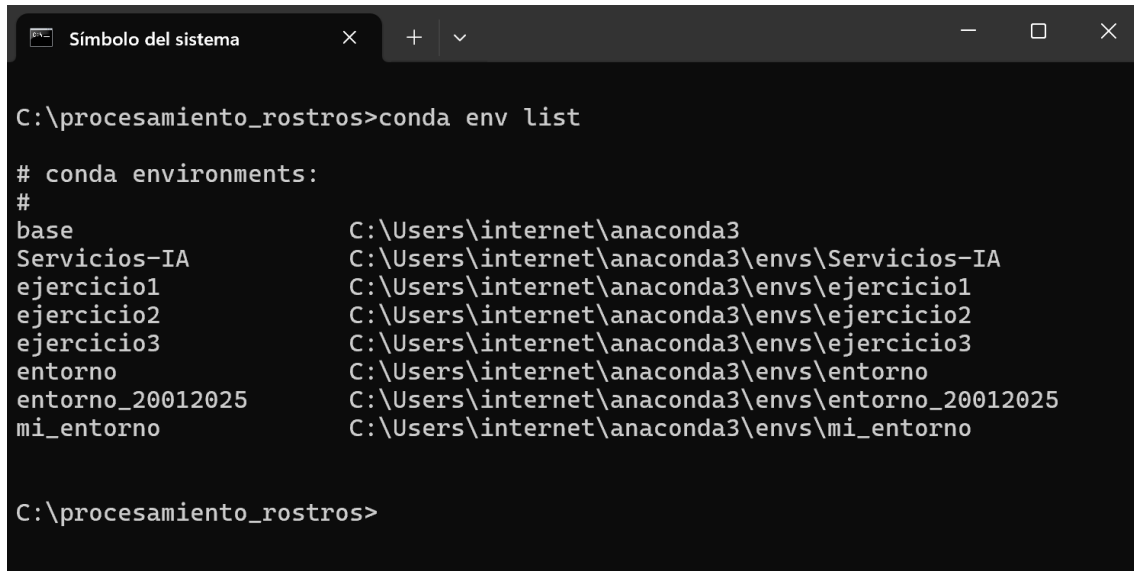
9.7.- Directorio C:\Procesamiento_Rostros\Source_Ejercicio003

- **Ejercicio003.ipynb (9.308 bytes)**: Notebook de Jupyter con el código y explicaciones del tercer ejercicio, centrado en la comparación de imágenes de coches.

9.8.- Resumen

Este conjunto de ficheros y directorios proporciona un entorno completo para la enseñanza y el análisis de técnicas de procesamiento de imágenes, con un enfoque en el reconocimiento facial. Se incluyen notebooks de Jupyter con ejercicios, imágenes de entrada y salida, y herramientas necesarias para la configuración del entorno de trabajo. Además, se proveen enlaces a software esencial como Anaconda, CMake y Visual Studio Code para garantizar la correcta ejecución del material entregado.

10.- INSTALACIÓN DEL ENTORNO A OTRO PC.

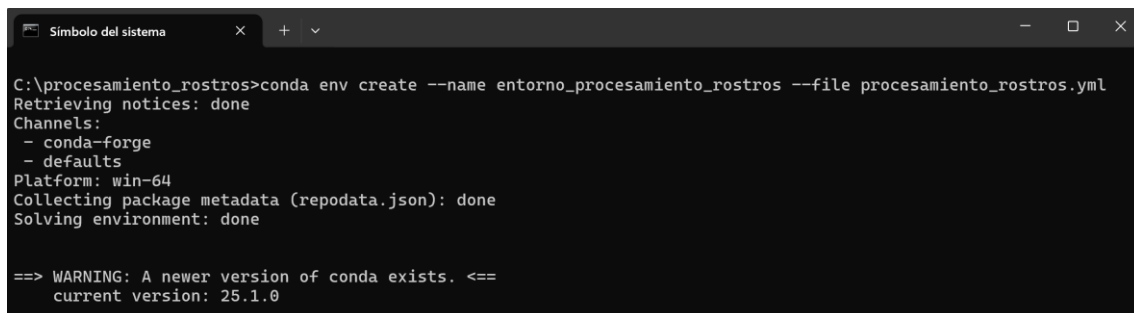


```
C:\procesamiento_rostros>conda env list

# conda environments:
#
base                        C:\Users\internet\anaconda3
Servicios-IA                C:\Users\internet\anaconda3\envs\Servicios-IA
ejercicio1                  C:\Users\internet\anaconda3\envs\ejercicio1
ejercicio2                  C:\Users\internet\anaconda3\envs\ejercicio2
ejercicio3                  C:\Users\internet\anaconda3\envs\ejercicio3
entorno                     C:\Users\internet\anaconda3\envs\entorno
entorno_20012025            C:\Users\internet\anaconda3\envs\entorno_20012025
mi_entorno                  C:\Users\internet\anaconda3\envs\mi_entorno

C:\procesamiento_rostros>
```

En esta pantalla, se muestra la lista de entornos disponibles ejecutando el comando `"conda env list"` y presionando la tecla Intro.



```
C:\procesamiento_rostros>conda env create --name entorno_procesamiento_rostros --file procesamiento_rostros.yml
Retrieving notices: done
Channels:
- conda-forge
- defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
current version: 25.1.0
```

En esta pantalla, se duplica el entorno de desarrollo en otro sistema utilizando el comando `"conda env create -name entorno_procesamiento_rostros -file procesamiento_rostros.yml"`, que lee las especificaciones del archivo YAML para reconstruir el ecosistema de trabajo con todas sus dependencias y configuraciones, garantizando coherencia y facilitando el trabajo en equipo en proyectos de software.

```

Simbolo del sistema - conda x + v - □ x
C:\procesamiento_rostros>conda activate entorno_procesamiento_rostros
(entorno_procesamiento_rostros) C:\procesamiento_rostros>

```

En esta pantalla, se activa el entorno con el comando "`conda activate entorno_procesamiento_rostros`" y se presiona la tecla Intro para ejecutarlo.

11.- MAPA MENTAL



El mapa mental organiza el desarrollo de un proyecto de procesamiento de imágenes y reconocimiento facial en Python, detallando objetivos, herramientas, configuración del entorno, ejercicios prácticos, pruebas, conclusiones, bibliografía y anexos, con enfoque en el análisis de emociones, difuminado de rostros y comparación facial.

ÍNDICE ALFABÉTICO

A

académicos	41
accesibles	29
acción	8
actividades	4
además	18, 25
adquisición	10
algoritmos	18, 23, 29, 33, 34, 38, 40, 42
almacenamiento	25
análisis	4, 5, 15, 16, 18, 19, 20, 23, 24, 25, 28, 32, 39, 42, 43, 44, 47
anexos	47
ángulos	29, 30, 35
aprendizajes	34, 36, 38
archivos	25, 42
áreas	34
arreglos	5
artículos	39, 41
aspectos	24, 34, 36, 38
ausencia	33
automática	10
automático	4, 5, 39
avanzadas	4, 24

B

barras	5
basadas	4
bases	16, 20, 38, 40
bibliografía	39, 41, 47
biblioteca	9, 24, 39, 40, 41
bibliotecas	9, 10, 11, 16, 20, 25, 37, 39, 41, 42
bytes	42, 43, 44

C

cálculo	40
calidad	18, 23, 28, 32, 36
cambios	19
cantidad	24
capacidad	31, 36
características	24, 29, 32, 37, 39, 40
caras	16, 17, 18, 20
casos	4, 30, 31, 32, 35, 37
celdas	14
científico	40
claras	30
claros	31
cmake	2, 5, 6, 9, 43
coches	44
códigos	4, 34
coherencia	45
colección	41
comentarios	19
complejos	38
complementarios	5
comportamiento	15, 24
comprensión	4, 36, 38, 40, 41
comunidad	41
conceptos	39
conclusiones	47
conda	8, 9, 12, 45, 46
conjunto	29, 39, 42, 44
consistentes	35
consola	12, 27, 32
continuación	6, 7, 42
creación	14, 39
crítica	34, 36

cruciales	41
cuatros	28
cubiertos	37
cuyas	19

D

dependencias	5, 10, 42, 45
depuración	14
desafíos	34, 36, 38, 41
desarrolladores	41
desarrollo	4, 5, 6, 10, 13, 34, 36, 37, 38, 39, 40, 41, 43, 45, 47
desarrollos	38
descripción	15
detalle	15, 42
detección	5, 18, 19, 24, 25, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41
difíciles	30
dificultades	36
dinámica	41
directorios	2, 7, 42, 44
distintos	29
distribución	42
diversos	36, 40
dlib	5, 9, 39, 41
documentación	4, 41
dominantes	19

E

editor	43
educativos	42
efectividad	18, 24, 38
eficaz	37
ejecución	5, 8, 10, 30, 31, 34, 36, 37, 42, 43, 44
ejemplos	40, 41
ejercicio	15, 19, 24, 29, 43, 44
elección	35, 37
elementos	40
emoción	15, 21, 23
enfoque	23, 24, 37, 40, 44, 47
enlaces	43, 44
enojo	18, 23
entorno	2, 3, 4, 6, 7, 8, 9, 10, 13, 37, 42, 43, 44, 45, 46, 47
errores	29, 30, 34, 35
escenarios	24, 38
esenciales	7, 16, 20, 41
especificaciones	45
específicos	41
estabilidad	37
estrategias	29
estudios	15, 24
etapas	44
éticos	24
etiquetas	14, 21, 23
evaluación	29
experiencia	36
explicaciones	44
expresión	22
extensiones	2, 13, 14
extracción	25, 32

F

face	4, 5, 16, 24, 25, 31, 35, 37, 39, 40, 41
faciales	7, 23, 24, 25, 28, 29, 30, 32, 37, 39, 40
fáciles	22
factores	37
fallos	31, 33

Comparación de Rostros Utilizando Reconocimiento Facial en Python

fases	7
felicidad	18, 23
fiabilidad	35
ficheros	42, 44
flask	5
flujo	33
formatos	40
fuentes	39, 41
función	12, 27, 30, 32
funcionales	33
funcionalidad	23, 29, 38
funcionalidades	4, 5
funcionamiento	29
fundamentales	40
fundamentos	41
futuras	34
futuros	36, 38

G

generación	5, 7, 8
gestión	5, 7
gráficas	5, 39
gráficos	7, 14, 39

H

hallazgos	34
herramientas	5, 15, 24, 29, 35, 37, 39, 41, 43, 44, 47
histórica	41
humanos	4

I

iguales	27, 30, 32
iluminación	18, 23, 29, 35, 38
imágenes	4, 5, 7, 15, 16, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 47
implementación	4, 31, 33, 41, 42
implementaciones	34
importancia	37
instalación	2, 5, 6, 7, 8, 9, 10, 13, 14
instrucciones	6
intactos	22
integración	10
inteligencia	4, 5
interactivos	14
interrupciones	33
investigación	39

L

largas	5
lecciones	34, 36, 37
lenguaje	14
lento	18, 24, 36
librería	9, 12, 39, 40
licencia	13
limitaciones	18, 23, 29
lista	12, 45
logros	34
lotes	10

M

manipulación	5, 31, 40
manipulaciones	5
mapa	47
mapas	14
matemáticas	5, 20, 40
matemático	32
materiales	39
matrices	40

mecánica	37
mensajes	21, 31, 33, 35
metodología	29
métodos	4, 40
modelos	18, 23, 24
modificación	19
módulos	31
multidimensionales	40
múltiples	14, 18, 30, 31, 33, 36, 38

N

necesaria	5
necesarios	5, 24
necesidades	37
neuronales	5, 40, 41
normativas	23
numéricos	16, 24

O

objetivos	34, 47
objetos	39
ocultos	18, 23
oficiales	39
opencv	5
operaciones	5, 20, 25, 40
oportunidad	38
organización	42
orientación	37
originales	30

P

pantalla	7, 8, 9, 10, 15, 17, 18, 22, 45, 46
paquetes	5, 42
paralelo	33
pasos	13, 29
pdf	42
perspectivas	41
píxeles	36
posibles	5, 28
práctica	4, 40
práctico	24, 40
precisa	35
precisas	36
precisión	18, 23, 29, 32, 34, 41
precisos	18, 24, 33
presentaciones	14
presentes	4, 15, 20, 27, 36
preservación	15
principales	34
prioridad	36
privacidad	15, 18, 23
problemas	9, 32, 41
procesos	40
programación	5, 41
propuesto	7
protección	23
proyectos	45
pruebas	29, 32, 33, 34, 47
públicos	24
pulsación	8, 9, 10
python	5

R

recognition	4, 5, 16, 24, 25, 31, 35, 37, 39, 40, 41
reconocimiento	4, 5, 16, 20, 23, 24, 25, 27, 28, 29, 30, 34, 36, 37, 38, 39, 40, 41, 42, 44, 47
rectángulos	21
recurso	40, 41
redes	5, 40, 41
referencia	3, 40
reflexión	34
relación	41

Comparación de Rostros Utilizando Reconocimiento Facial en Python

relevantes	39
renderización	14
rendimiento	24, 29, 33, 36, 37, 38
repositorio	10, 41
representación	35
representaciones	39
requisitos	12, 33, 38
resolución	30, 32, 33, 34, 36, 37, 38, 41
resultante	27, 43
retos	36, 37
reunión	20
reuniones	44
revisión	41
robusto	30, 33, 36, 37, 38
rostros	4, 5, 7, 8, 15, 16, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47
rutas	27

S

satisfactorios	33
sección	29, 32, 34
secuencia	7
similitudes	4
situaciones	35, 36
sombreros	36
source	7

T

tareas	5, 34
tecla	8, 9, 10, 23, 45, 46
teclas	14
técnicas	4, 18, 24, 29, 37, 39, 40, 41, 44
técnicos	24, 41
tecnologías	5, 41

términos	13
trabajos	36, 37, 38
transformaciones	39, 40
tratamiento	4

U

ubicación	32
unidades	24
usuario	23, 35
utilización	7

V

variables	24
variaciones	29
variedad	39, 40
vectores	24, 40
velocidad	34
versatilidad	5
versión	12, 13
videos	16, 18, 24
virtuales	5
visibles	37
visión	16, 18, 23, 24, 29, 38, 39, 40, 41, 42
visuales	5
visualización	5, 24, 25, 31, 33, 37
visualizaciones	5, 39
volúmenes	36, 38

W

windows	43
---------	----