

DEPARTAMENTO: Ciencias de la Ingeniería **PROFESOR:** Mg. Gustavo Guaigua A.

CARRERA: Sistemas de Información

ESTUDIANTES: Cando Orlando, Columba Ámbar, Guamba Cristian, Tipán Gissela

CURSO: 9 **PARALELO:** "B"

DESCRIPCIÓN: Proyecto OpenCV.

ASIGNATURA: Tendencias innovadoras de la profesión **SEMESTRE:** 2024 B

TEMA: Reconocimiento facial con Python

El presente proyecto abarca el registro de rostros para entrenar un algoritmo de reconocimiento facial.

Enlace del proyecto:

<https://github.com/Cristiangpbf/proyecto-open-cv.git>

Parte 1: Almacenar rostros

El siguiente código realiza un reconocimiento facial en tiempo real usando la cámara de un dispositivo para capturar imágenes de una persona, recortar las caras detectadas y guardarlas en una carpeta específica.

Explicación del código

- **Importación de bibliotecas**

```
almacenando_rostros.py x
1 import cv2
2 import os
3 import imutils
```

- o **cv2:** Importa OpenCV, una biblioteca para procesamiento de imágenes y visión por computadora.
- o **os:** Permite interactuar con el sistema operativo, en este caso para crear carpetas y verificar rutas.
- o **imutils:** Proporciona funciones adicionales de procesamiento de imágenes, como redimensionar imágenes.

- **Definición de variables**

```
5 personName = 'Cristian'
6 dataPath = 'C:/imagenes_reconocimiento_facial/Data'
7 personPath = dataPath + '/' + personName
```

- o **personName:** Es el nombre de la persona a la que se le van a capturar las imágenes (en este caso, 'Cristian').
- o **dataPath:** Ruta donde se guardarán las imágenes. Se establece como 'C:/imagenes_reconocimiento_facial/Data'.
- o **personPath:** Ruta completa que incluirá el nombre de la persona, lo que sirve para crear una carpeta personalizada para guardar las imágenes.

- **Crear la carpeta para almacenar las imágenes**

```

9      if not os.path.exists(personPath):
10         print('Carpeta creada: ', personPath)
11         os.makedirs(personPath)

```

- o Verifica si la carpeta correspondiente a la persona (personPath) ya existe. Si no existe, la crea.
- o La función os.makedirs() crea todas las carpetas necesarias en la ruta.

- **Configuración de la cámara**

```

13     cap = cv2.VideoCapture(1, cv2.CAP_DSHOW)
14     # Descomentar para capturar registros de un video.
15     # cap = cv2.VideoCapture('Video.mp4')

```

- o **cap:** Abre la cámara del dispositivo (en este caso, el índice 1 para seleccionar la cámara secundaria). La opción cv2.CAP_DSHOW se usa para utilizar DirectShow, que es compatible con muchas cámaras en Windows. Si se desea usar un video en lugar de la cámara, se puede descomentar la línea cap = cv2.VideoCapture('Video.mp4').

- **Cargar el clasificador de rostros e inicializar contador para ocurrencias**

```

17     faceClassif = cv2.CascadeClassifier(cv2.data.harcascades +
18                                         'haarcascade_frontalface_default.xml')
19     count = 0

```

- o **faceClassif:** Carga el clasificador de rostros de OpenCV, basado en el archivo haarcascade_frontalface_default.xml. Este clasificador permite detectar rostros en imágenes mediante el algoritmo Haar Cascade.
- o **count:** Es un contador que se usará para nombrar las imágenes de los rostros capturados, asegurando que cada archivo tenga un nombre único.

- **Bucle principal**

```

21     while True:
22         ret, frame = cap.read()
23         if not ret: break

```

- o **while True:** Este es un bucle infinito que se ejecuta hasta que el usuario decida detenerlo o se hayan capturado 300 imágenes.

- o **ret, frame = cap.read():** Captura una imagen de la cámara. Si ret es False, significa que no se pudo capturar la imagen, y el bucle termina.

- **Preprocesamiento de la imagen**

```
25     frame = imutils.resize(frame, width=640)
26     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
27     auxFrame = frame.copy()
```

- o **frame = imutils.resize(frame, width=640):** Redimensiona la imagen capturada para reducir su tamaño a un ancho de 640 píxeles.
- o **gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY):** Convierte la imagen a escala de grises, ya que el clasificador de rostros trabaja mejor con imágenes en blanco y negro.
- o **auxFrame = frame.copy():** Hace una copia de la imagen original que se usará más adelante para recortar y guardar los rostros.

- **Detección de rostros**

```
29     faces = faceClassif.detectMultiScale(gray, scaleFactor: 1.3, minNeighbors: 5)
```

- o **faces:** Usa el clasificador de rostros para detectar las caras en la imagen gray. La función detectMultiScale devuelve una lista de coordenadas de las caras detectadas. Los parámetros 1.3 y 5 controlan la escala y el número mínimo de vecinos requeridos para una detección.

- **Recorte y almacenamiento de las caras**

```
31     for (x, y, w, h) in faces:
32         cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
33         rostro = auxFrame[y:y + h, x:x + w]
34         rostro = cv2.resize(rostro, dsize=(150, 150), interpolation=cv2.INTER_CUBIC)
35         cv2.imwrite(personPath + '/rostro_{}.jpg'.format(count), rostro)
36         count = count + 1
```

- o **for (x, y, w, h) in faces:** Recorre cada cara detectada. Las variables (x, y) representan las coordenadas de la esquina superior izquierda de la cara, mientras que w y h son el ancho y la altura de la cara.
- o **cv2.rectangle:** Dibuja un rectángulo verde alrededor de cada rostro detectado.
- o **rostro = auxFrame[y:y + h, x:x + w]:** Recorta la cara de la imagen original.
- o **rostro = cv2.resize(rostro, (150, 150), interpolation=cv2.INTER_CUBIC):** Redimensiona el rostro recortado a 150x150 píxeles para estandarizar el tamaño.
- o **cv2.imwrite(personPath + '/rostro_{}.jpg'.format(count), rostro):** Guarda la imagen recortada del rostro con un nombre único basado en el contador count.
- o **count = count + 1:** Incrementa el contador para el siguiente rostro.

- **Mostrar la imagen**

```
38     cv2.imshow( winname: 'frame', frame)
```

- o `cv2.imshow`: Muestra la imagen procesada con los rectángulos dibujados alrededor de las caras en una ventana llamada `frame`.

- **Control de terminación**

```
40     k = cv2.waitKey(1)
41     if k == 27 or count >= 300:
42         break
```

- o **`cv2.waitKey(1)`**: Espera una tecla durante 1 milisegundo. Si se presiona la tecla Esc (código 27), el bucle se detendrá.
- o **`count >= 300`**: Si se han capturado 300 imágenes, el bucle también se detiene.

- **Liberar recursos y cerrar ventanas**

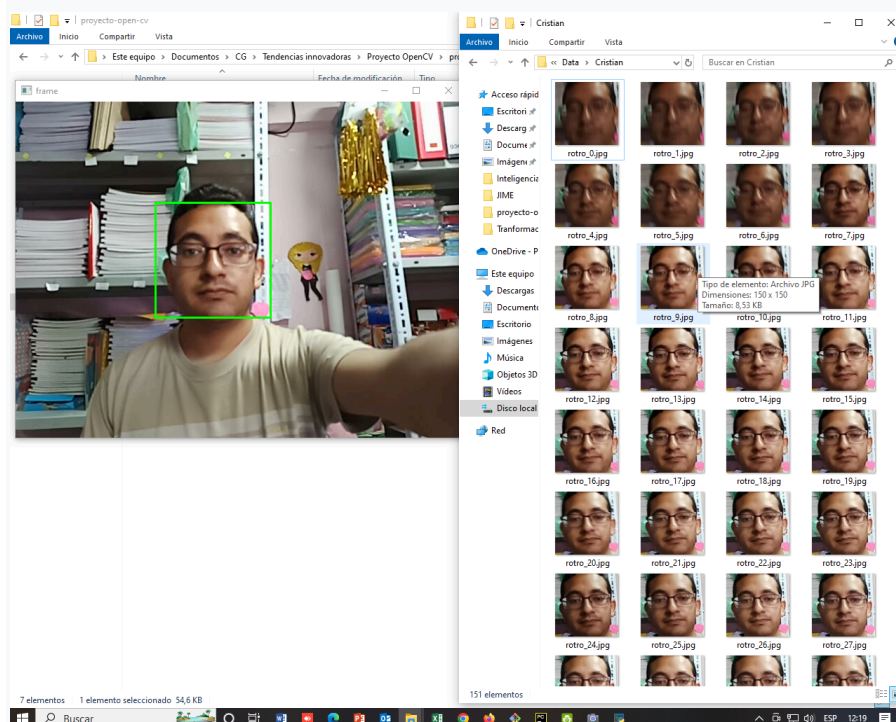
```
44     cap.release()
45     cv2.destroyAllWindows()
```

- o **`cap.release()`**: Libera la cámara una vez que el bucle ha terminado.
- o **`cv2.destroyAllWindows()`**: Cierra todas las ventanas de OpenCV.

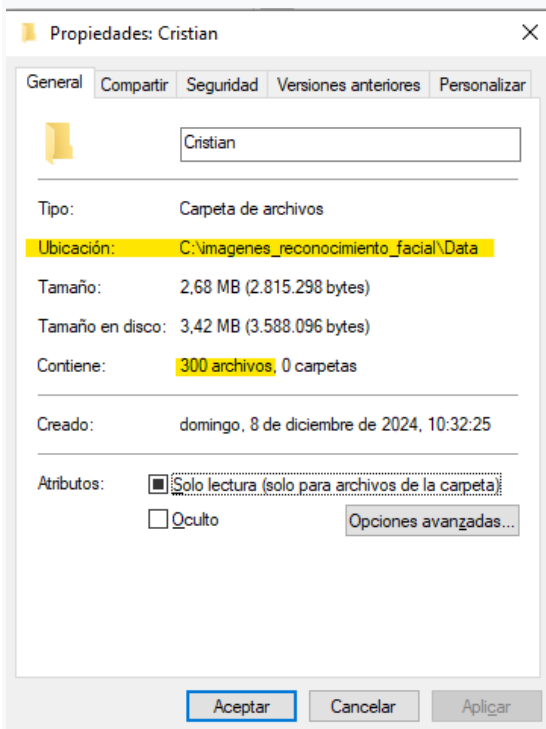
Aplicación del código

Se realiza la ejecución del código adaptando la cámara de un dispositivo Android para que Windows la reconozca como una cámara web mediante la aplicación DroidCam para Android y su cliente Windows.

Se ejecuta el código Python explicado anteriormente y se observa el registro de video desde la cámara web, así como el registro de los fotogramas en la carpeta asociada al nombre actual.



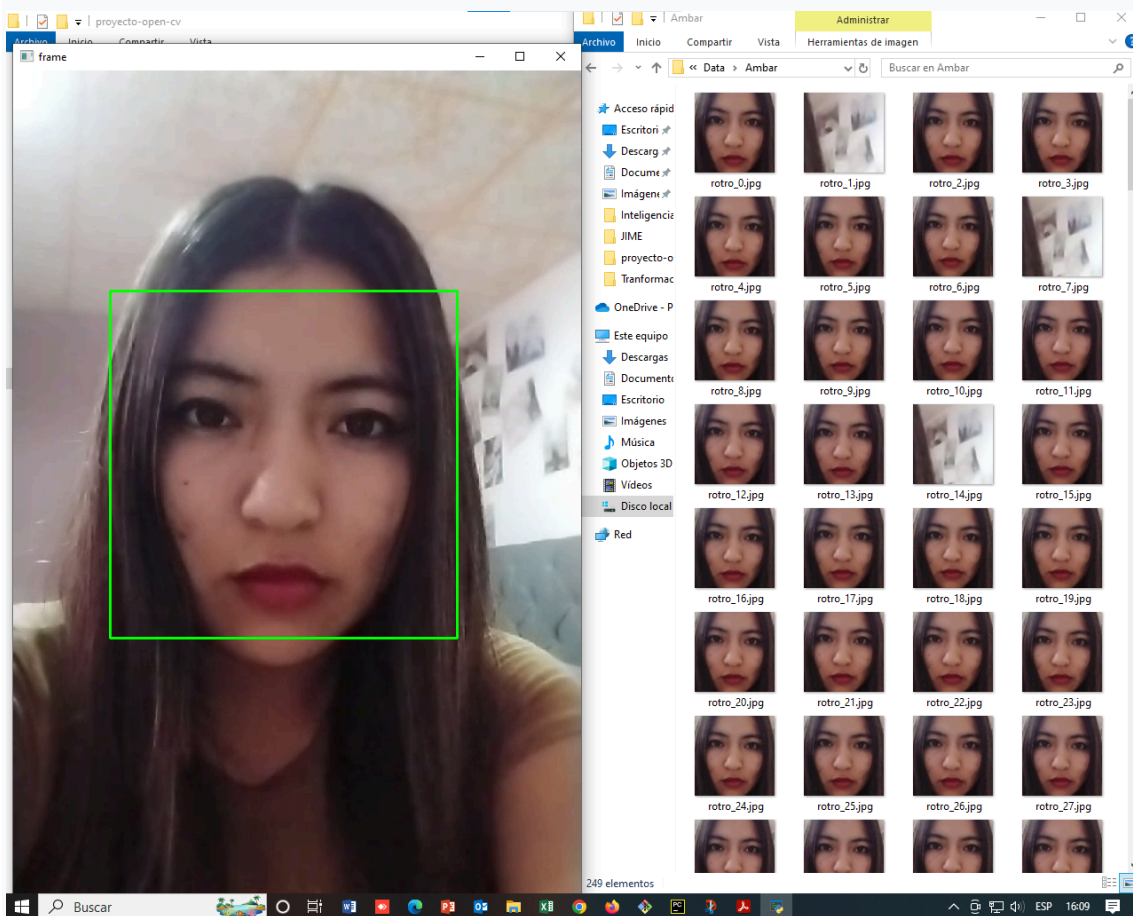
Se observa la creación del directorio con los 300 ficheros de los rostros identificados.



También se realiza la lectura desde un video en formato MP4. Para esto se cambia el origen del video.

```
17 # cap = cv2.VideoCapture(1, cv2.CAP_DSHOW)
18 # Descomentar para capturar registros de un video.
19 cap = cv2.VideoCapture('C:/imagenes_reconocimiento_facial/Videos/'+personName+'.mp4')
```

Al ejecutar el programa detecta el origen de video el fichero MP4 y almacena las imágenes de rostro identificados.



En este caso, algunos registros indican falsos positivos. Al tratarse de un volumen bajo de imágenes, se procedió a eliminar manualmente estos casos.

Parte 2: Entrenamiento

En esta etapa se realiza el proceso de entrenamiento mediante varios métodos para entrenar el reconocedor.

Explicación del código

- **Carga de datos.**

```
5 dataPath = 'recursos_reconocimiento_facial/Data' #Cambia a la ruta donde hayas almacenado Data
```

- o Especifica la carpeta Data donde están organizadas las imágenes. Esta carpeta debe tener una estructura como esta.

```

Data/
├── Persona1/
│   ├── rostro1.jpg
│   ├── rostro2.jpg
│   └── ...
├── Persona2/
│   ├── rostro1.jpg
│   ├── rostro2.jpg
│   └── ...

```

- **Lista de personas.**

```

6 peopleList = os.listdir(dataPath)
7 print('Lista de personas: ', peopleList)

```

- o Especifica la carpeta Data donde están organizadas las imágenes. Esta carpeta debe tener una estructura como esta.

- **Carga de datos.**

```

5 dataPath = 'recursos_reconocimiento_facial/Data' #Cambia a la ruta donde hayas almacenado Data

```

- o Lee las subcarpetas dentro de Data para obtener la lista de personas (clases del modelo). Por ejemplo: ['Persona1', 'Persona2'].

- **Inicialización.**

```

9 labels = []
10 facesData = []
11 label = 0

```

- o **labels:** Almacena las etiquetas numéricas asociadas a cada persona.
- o **facesData:** Contiene las imágenes en formato de escala de grises.
- o **label:** Asigna un número único a cada persona (0 para la primera, 1 para la segunda, etc.).

- **Recorrido por las carpetas de personas.**

```

13 for nameDir in peopleList:
14     personPath = dataPath + '/' + nameDir
15     print('Leyendo las imágenes')

```

- o Se recorre cada carpeta dentro de Data.

- **Cargar imágenes de cada persona.**

```

17 for fileName in os.listdir(personPath):
18     print('Rostros: ', nameDir + '/' + fileName)
19     labels.append(label)
20     facesData.append(cv2.imread(personPath + '/' + fileName, 0))

```


- o Se carga cada archivo de imagen en escala de grises (cv2.imread(..., 0)).
- o Las imágenes se almacenan en facesData.
- o La etiqueta numérica correspondiente a la persona se guarda en labels.

- **Incrementar etiqueta.**

```
24     label = label + 1
```

- o Después de procesar todas las imágenes de una persona, se incrementa la etiqueta para la siguiente.

- **Estadísticas.**

```
26     print('labels= ', labels)
27     print('Número de etiquetas 0: ', np.count_nonzero(np.array(labels) == 0))
28     print('Número de etiquetas 1: ', np.count_nonzero(np.array(labels) == 1))
```

- o Se imprimen las etiquetas generadas y la cantidad de imágenes asociadas a cada etiqueta.

- **Creación del reconocedor.**

```
30     # Métodos para entrenar el reconocedor
31     face_recognizer = cv2.face.EigenFaceRecognizer_create()
32     # face_recognizer = cv2.face.FisherFaceRecognizer_create()
33     # face_recognizer = cv2.face.LBPHFaceRecognizer_create()
```

- o Se pueden utilizar varios métodos para el reconocedor.
 - EigenFaces
 - FisherFaces
 - LBPH (Local Binary Patterns Histograms)

Cada método tiene sus características:

1. **EigenFaces:** Bueno para condiciones controladas, pero sensible a variaciones como iluminación.
2. **FisherFaces:** Mejor manejo de variaciones entre clases, pero requiere más datos.
3. **LBPH:** Más robusto frente a cambios en iluminación y expresión.

- **Entrenamiento.**

```
35     # Entrenando el reconocedor de rostros
36     print("Entrenando...")
37     face_recognizer.train(facesData, np.array(labels))
```

- o **facesData:** Imágenes de rostros en escala de grises.
- o **labels:** Etiquetas asociadas a cada rostro.

- **Guardar modelo.**





```
39 # Almacenando el modelo obtenido
40 face_recognizer.write('modeloEigenFace.xml')
41 #face_recognizer.write('modeloFisherFace.xml')
42 # face_recognizer.write('modeloLBPHFace.xml')
43 print("Modelo almacenado...")
```

- o El modelo entrenado se guarda en un archivo .xml para que pueda ser cargado y usado en la etapa de predicción.
- o Dependiendo del método de entrenamiento utilizado, se cambia el nombre del archivo guardado:
 - modeloEigenFace.xml
 - modeloFisherFace.xml
 - modeloLBPHFace.xml

Aplicación del código

Se realiza la ejecución del código para cada método de entrenamiento. Se generan 3 diferentes archivos xml con los resultados del análisis de los rostros registrados.

Como resultado final de la ejecución del script Python se generan los 3 archivos xml. Uno por cada método de entrenamiento utilizado.

```
☐  entrenandoRF.py
☐  modeloEigenFace.xml
☐  modeloFisherFace.xml
☐  modeloLBPHFace.xml
```