RECONOCIMIENTO FACIAL

PROGRAMA DE RECONOCIMIENTO FACIAL EN TIEMPO REAL USANDO PYTHON

Pablo García Fernández
Periféricos y Dispositivos de Interfaz Humana

UNIVERSIDAD DE GRANADA

INTRODUCCIÓN

El objetivo del proyecto es realizar una demostración de un programa de reconocimiento facial en tiempo real a través de una webcam utilizando Python.

Las librerías que se utilizan son OpenCV, librería de visión artificial; y Face Recognition, librería para reconocimiento facial. Ambas están disponibles tanto en Github como a través del instalador *pip*.

En este documento se explicará la diferencia entre detección facial y reconocimiento facial, el desarrollo de un programa de detección y otro de reconocimiento.

DETECCIÓN Y RECONOCIMIENTO FACIAL

Antes de realizar ambos programas es importante conocer que es detección facial y que es reconocimiento facial.

Se conoce como detección facial a las técnicas que, utilizando una inteligencia artificial entrenada, determinan la aparición de rostros humanos en las imágenes. Se utilizan algoritmos capaces de analizar y separar las caras del resto de información presente.

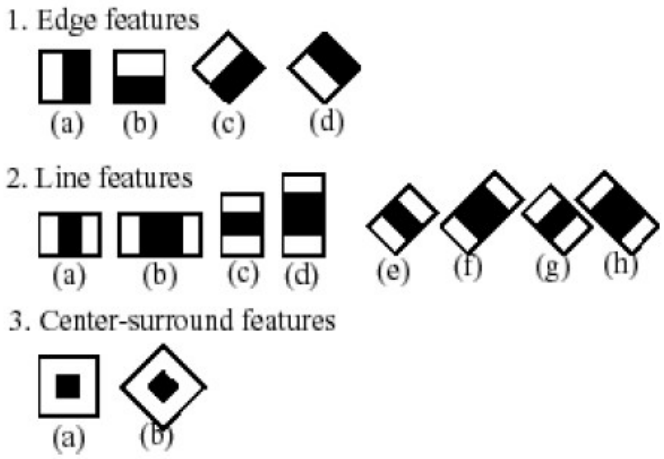
El reconocimiento facial no solo es capaz de detectar los rostros si no que, a través de tecnología biométrica, también determina la identidad de la persona.

	Detección	Reconocimiento facial
Encuentra el rostro	✓	✓
Identifica rasgos faciales	✓	✓
Funciona para imágenes y videos	✓	✓
Utiliza datos biométricos	✗	✓
Vincula los rasgos faciales con un nombre	✗	✓

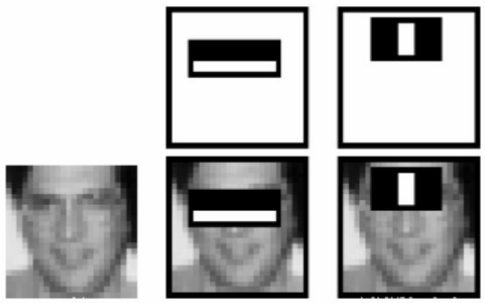
DETECCIÓN FACIAL CON OPENCV

El programa que se describe a continuación tiene como propósito realizar detección facial a través de la webcam del ordenador ayudándose de la biblioteca OpenCV.

Antes de ir paso a paso sobre el código, hay que explicar el algoritmo que se utiliza para la detección facial. OpenCV utiliza el algoritmo de Viola & Jones para detectar rostros, pero también es posible detectar objetos como coches, peatones, etc. Para realizar la detección se entrena un clasificador en cascada, es decir, se aplican una serie de clasificadores sobre una región de interés donde puede hallarse el objeto en busca de ciertas características. Estas características se llaman Haar-Like Features y tienen la siguiente forma:



Al entrenar el clasificador se buscan combinaciones de estas características en las imágenes de ejemplo y se eligen las más significativas para que formen parte del clasificador. Cuando todos los clasificadores dan positivo, el algoritmo considera que hay una coincidencia y que se ha encontrado el objeto deseado.



Para el programa de detección facial se utiliza el clasificador en cascada que proporciona OpenCV.

Los pasos preliminares son cargar archivo XML que utilizará el clasificador para entrenar la detección de rostros e iniciar la webcam.

```
import cv2

#Se carga el clasificador Haar (https://github.com/opencv/opencv/tree/master/data/haarcascades)
cascadeClassifier = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')

#Se inicia la webcam
webcam = cv2.VideoCapture(0)
```

A continuación, el programa establece un bucle infinito en el cual se capturara cada frame de la webcam.

```
while(1):
    #Se captura un frame de la webcam para analizar
    valid, img = webcam.read()

    if valid:
        #Se convierte la imagen a escala de grises
        img_gris  = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

En caso de que se haya capturado correctamente se continúa el programa y se convierte la imagen a escala de grises. Esto se hace porque la función *detectMultiScale()* de OpenCV requiere ese formato.

```
#Usando el clasificador se detectan las caras
coord_caras = cascadeClassifier.detectMultiScale(
    img_gris,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(30, 30),
    #flags=cv2.cv.CV_HAAR_SCALE_IMAGE
    )
```

Ahora se procede a pintar un recuadro rojo sobre cada cara encontrada por el algoritmo iterando sobre el resultado.

```
#Cada cara encontrada se va a marcar con un recuadro rojo
for(x,y,w,h) in coord_caras:
    cv2.rectangle(img,  (x, y), (x+w, y+h),(0,0,255),3)

#En una ventana nueva se muestra el resultado
cv2.imshow('Webcam', img)
```

Por último, se establece la tecla para terminar el bucle infinito y poder cerrar el programa.

```
        #Se establece la tecla ESC como condición de parada del programa
        key = cv2.waitKey(5) & 0xFF
        if key == 27:
            cv2.destroyAllWindows()
            break

webcam.release()
```

El resultado al ejecutar el programa y detectar un rostro es el siguiente:

RECONOCIMIENTO FACIAL

Para realizar el reconocimiento facial, además de OpenCV, es utiliza una biblioteca llamada Face Recognition. Esta biblioteca aporta funciones como *face_locations()* que nos permite realizar la detección facial de forma sencilla.

El programa funciona realiza los reconocimientos a partir de una serie de fotos de las que se extraen los encodings para guardarlos, se comparan los encodings encontrados con los conocidos y se asigna el rostro con la mejor coincidencia.

Lo primero es cargar las imágenes, extraer sus encodings y asignarles los nombres correspondientes.

```
#Se cargan la imagen o imagenes
imagen1 = face_recognition.load_image_file('./caras/Pablo.jpg')
imagen2 = face_recognition.load_image_file('./caras/chalamet.jpg')

#Extrae los encodings de los rostros
encoding1 = face_recognition.face_encodings(imagen1)
encoding2 = face_recognition.face_encodings(imagen2)

#Se define un array con los encodings y otro con los nombres
encodings_conocidos = np.array([
    encoding1,
    encoding2
])

nombres_conocidos = np.array([
    "Pablo Garcia",
    "Timothee Chalamet"
])
```

Al igual que con la detección facial, se hace un bucle infinito para ir capturando frame a frame la webcam. Se definen las variables que se van a utilizar en el bucle y que tienen que ser redeclaradas en cada iteración.

```
while(1):
    #Definimos arrays y variables a utilizar
    coord_caras = np.array([]) #Coordenadas de las caras detectadas
    encodings_caras = np.array([]) #Encodings de las caras detectadas
    nombres_caras = [] #Nombre de las personas
    total_distances = np.array([100.0,100.0,100.0,100.0,100.0,100.0]) #Array de "distancias" entre rostros
    procesar = True
    nombre = "Desconocido" #Variable para el nombre

    valid, img = webcam.read()
```

Si se captura correctamente la imagen de la webcam, se convierte a formato RGB y se aplica una reducción en caso de que se quiera aumentar la velocidad de la detección. Con *face_locations()* y *face_encodings* se realiza la detección facial y se obtienen los encodings para las caras encontradas.

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #Convertir de BGR a RGB
img_rgb = cv2.resize(img_rgb, (0,0), fx = 1.0/reduccion, fy= 1.0 /reduccion) #Se aplica la reducción
#Se localizan las caras y se obtienen sus encodings
coord_caras = face_recognition.face_locations(img_rgb)
encodings_caras = face_recognition.face_encodings(img_rgb, coord_caras)
```

Para cada cara encontrada en la imagen se compara con los encodings de las caras que se pretenden reconocer y se calculan las distancias entre ellas. Las distancias serán utilizadas para establecer cual es la mejor coincidencia, cuanto menor la distancia mejor la coincidencia.

```
for cara in encodings_caras:
    #Se buscan coincidencias con los encoding conocidos
    coincidencias = face_recognition.compare_faces(encodings_conocidos, cara, 0.2)
    matches = np.array(coincidencias)  #Se convierte en un array de numpy para poder
    face_distances = face_recognition.face_distance(encodings_conocidos, cara) #"Dis
    i=0
```

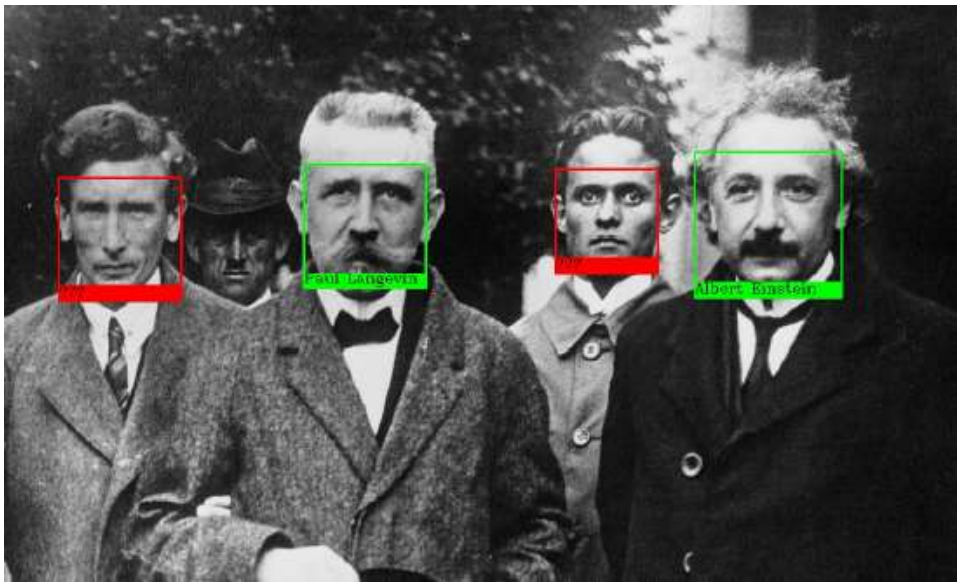
Si el rostro encontrado coincide con alguno conocido entonces se procede a calcular con cual de ellos se produce la mejor coincidencia. Esto se hace sumando las distancias totales y buscando la menor. Una vez calculada la mejor coincidencia se obtiene el nombre de la persona para mostrarse posteriormente.

```
if matches.all():
    #Para cada cara sumamos las "distancias"
    for face in face_distances:
        total_distances[i] = np.sum(face)
        i+=1
    best_match_index = np.argmin(total_distances) #Se la mejor coincidencia
    nombre = nombres_conocidos[best_match_index] #Se asigna el nombre de la mejor coincidencia
#Se añade al array de nombres
nombres_caras.append(nombre)
```

La diferencia con el programa anterior a la hora de marcar las caras es que si se ha producido un reconocimiento, el recuadro será de color verde y mostrará el nombre de la persona.

```
        #Recorremos tanto las coordenadas como los nombres
        for(top, right, bottom, left), nombre in zip(coord_caras, nombres_caras):
            top *= reduccion
            right *= reduccion
            bottom *= reduccion
            left *= reduccion

            #Cambia el color si es conocido o no
            if nombre == 'Desconocido':
                color = (0,0,255) #Color rojo
            else:
                color = (0,255,0) #Color verde

            #Se dibuja el rectángulo que indica la posición de la cara
            cv2.rectangle(img, (left,  top), (right, bottom), color, 2)
            cv2.rectangle(img, (left, bottom - 20), (right, bottom), color, -1)

            cv2.putText(img,nombre, (left, bottom -6), font, 0.6, (0,0,0), 1)

        #En una ventana nueva se muestra el resultado
        cv2.imshow('Webcam', img)

        #Se establece la tecla ESC como condición de parada del programa
        key = cv2.waitKey(5) & 0xFF
        if key == 27:
            cv2.destroyAllWindows()
            break
webcam.release()
```

Al igual que en el programa anterior se establece la tecla ESC para terminar el programa cuando se desee. Ya que el resultado del programa con la webcam es difícil de mostrar en una sola captura de pantalla, un ejemplo de como sería es el siguiente.



Los rostros conocidos se muestran en verde y los desconocidos en rojo.