

Reconocimiento Facial - Machine Learning

Estudiantes: Álvaro Cárdenas - Esteban Lagos

ARTICLE INFO

Keywords:
Inteligencia Artificial
Ensayo
Expectativas

ABSTRACT

El reconocimiento facial tiene una amplia variedad de aplicaciones para las que puede ser empleado, tanto para mejorar la seguridad o en aplicaciones industriales hasta puede utilizarse en dispositivos personales. En este trabajo de investigación se pretende estudiar una aplicación de reconocimiento facial, identificar los métodos que se emplearon y los resultados que se obtuvieron.

1. Introducción

En actualidad tenemos un gran y continuo desarrollo dentro de lo que son las tecnologías de automatización. Los sistemas automatizados se han ido utilizando a pasos agigantados en una gran cantidad de aplicaciones, ya sea en tanto los ámbitos industriales, como en el uso cotidiano. La biometría es uno de los campos que también ha experimentado avance tecnológico muy grande, estando cada vez más presente en el uso cotidiano actual, donde la importancia de la seguridad es uno de los factores clave. En este ámbito los sistemas de verificación biométrica están en continuo desarrollo, surgiendo sistemas de reconocimiento facial, huella dactilar, de voz o incluso de retina. En este ensayo, se va a realizar un estudio de algunas de las tecnologías utilizadas en el desarrollo de uno de los sistemas biométricos, los sistemas de reconocimiento facial. Estos sistemas emplean algoritmos para reconocer la identidad de personas que aparecen en imágenes o en videos a partir de sus características fisiológicas.

2. Desarrollo

2.1. Herramientas de desarrollo

2.1.1. Lenguaje de programación

El lenguaje de programación que se utilizó en el proyecto fue Python la versión 3.6, ya que este lenguaje cuenta con una variedad de librerías que facilitan el desarrollo del proyecto, haciendo más fácil la programación del reconocimiento facial y la inteligencia artificial.

2.1.2. Entorno de desarrollo

El entorno de desarrollo utilizado ha sido Spyder, disponible bajo la distribución multiplataforma Anaconda, la cual es libre y abierta. La versión de Anaconda empleada ha sido la 5.1, también por ser la última estable a inicios del proyecto.

2.1.3. Librerías utilizadas

Las librerías que se utilizaron son:

- NumPy (versión 1.14.15): representa el paquete de matemáticas de Python más importante ya que añade funcionalidades de vectores y matrices.
- OpenCV (versión 3.4.1): es una librería de código abierto

que aporta funciones de visión artificial y machine learning.

- Face Recognition (versión 1.2.2): se trata de una librería que aporta herramientas para reconocer y manipular caras de una forma muy sencilla.

2.2. Procesamiento del reconocimiento facial

El reconocimiento facial se basa en una búsqueda de patrones para posteriormente hacer un análisis. Esta búsqueda y análisis está constituida por 4 fases las cuales son: detección, preprocesado, extracción de características y comparación y clasificación.

La primera fase de detección facial consiste en localizar el rostro en la imagen. A continuación, se realiza un preprocesado a la imagen para alinear y normalizar los rostros, de manera que se obtengan unas características geométricas en común con todas las imágenes que se procesen. Luego, se lleva a cabo una extracción de características faciales para obtener información útil que será usada para distinguir unas caras de otras. Finalmente, el vector de características obtenido será comparado con la base de datos para decidir a quién pertenece el rostro.

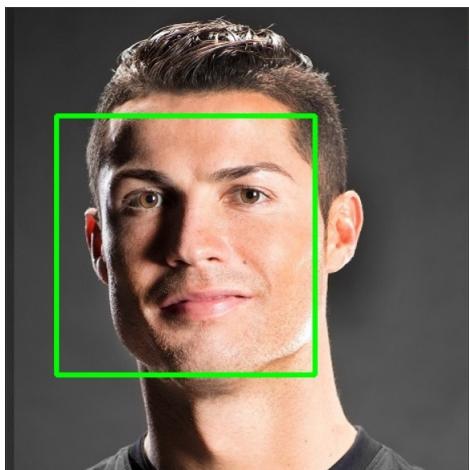
A continuación, veremos con más detalle cada una de las fases mencionadas anteriormente, y la gran importancia que tiene cada fase para el resultado final.

2.3. Detección facial

La detección facial es el proceso mediante el cual el sistema localiza la posición de los rostros de personas en una imagen o fotograma. La detección facial actualmente se hace de forma automática al contrario como se hacía en la antigüedad que era de forma manual. Los métodos de detección facial son numerosos y variados, pero veremos el método que se utilizó en el proyecto llamado “Detección de Haar-Cascade en OpenCV”.

Algoritmo de Viola-Jones El algoritmo que fue propuesto por Paul Viola y Michael Jones fue la primera técnica de detección facial que, además, presentaba una rapidez y precisión apta para su uso en tiempo real. Este método se basa en utilizar una serie de características denominadas Haar-like features, que son obtenidas a partir del producto escalar entre una imagen y un patrón simple del mismo tamaño que la imagen. Este producto escalar tendrá signo positivo

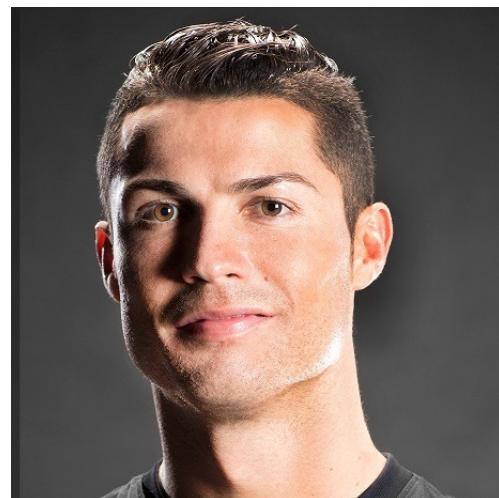
o negativo según cómo se describa el propio patrón, del que existe una gran variedad. Este resultado será el que se compare para determinar si se encuentra o no una cara en una zona determinada de la imagen. Dado un conjunto de Haar-like features y un conjunto de imágenes positivas y negativas, Viola y Jones implementaron una variante de AdaBoost para el entrenamiento de su clasificador. Este se trata de un algoritmo de aprendizaje creado por Freund y Schapire que consiste en obtener un clasificador fuerte a partir de varios clasificadores débiles que fijarán unos umbrales que, puestos en forma de cascada, formarán la región delimitada para una clase. Una vez haya sido entrenado este clasificador, será posible distinguir si el resultado de calcular las Haar-like features se corresponde con un rostro en cierto lugar de la imagen.



Detección de rostro

con sus respectivos puntos de referencia, haciendo uso de un algoritmo de aprendizaje automático llamado potenciación de gradiente.

Debido a que el entrenamiento de la cascada de regresores completa sería una tarea costosa tanto computacionalmente como en tiempo requerido, además de necesitar una considerable cantidad de imágenes para ello, la propia librería aporta el estimador ya entrenado para encontrar 68 puntos faciales. Cuando los puntos de referencia hayan sido localizados en la imagen, será una tarea sencilla realizar las transformaciones convenientes explicadas anteriormente, como la rotación o la inclinación para ajustar la cara a las condiciones necesarias. Cuando se lleven a cabo estas transformaciones en función de los puntos de la cara, se recortará la región donde se encuentra la cara para dejarla preparada para la extracción de las características.



(a) Original

2.4. Pre-procesamiento

Para poder ajustar la imágenes se hace uso de un algoritmo llamado estimación de punto de referencia que ha sido implementado en la librería Face Recognition, en el cual se busca estimar una serie de p puntos faciales de una forma eficiente, haciendo uso de una cascada de regresores. Estos puntos de referencia se denotan como:

$$S = (x_1^T, x_2^T, \dots, x_p^T)^T \in \mathbb{R}^{2p} \quad (1)$$

que recoge todas las coordenadas

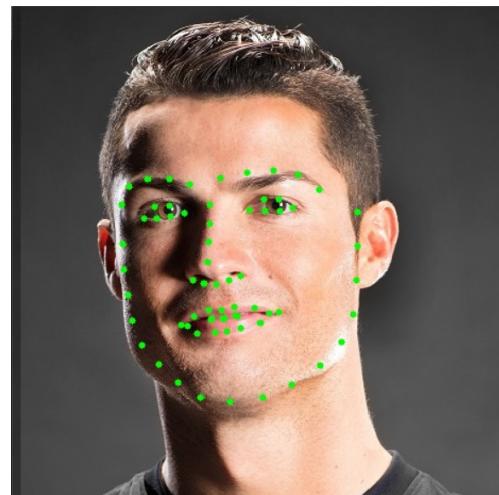
$$x_i = (x_i; y_i) \in \mathbb{R}^2 \quad (2)$$

de cada punto de referencia de la cara en una imagen I.

Este vector tendrá que ser estimado mediante una serie de regresores

$$r_t$$

,que deben ser entrenados en cascada para ser capaces de acercarse cada vez más a los diferentes puntos del rostro. Estos regresores son entrenados a partir de un conjunto de datos de entrenamiento que contiene un número de imágenes de caras



(b) 68 Puntos detectados

2.5. Extracción de características

Luego de obtener la imagen debidamente ajustada y centrada, la siguiente etapa consiste en obtener las medidas los rasgos faciales, es decir, obtener unos datos que describan de forma adecuada la cara, reduciendo la dimensionalidad de los datos de la imagen. Para este proyecto, se va a utilizar una red neuronal convolucional profunda la cual tienen como objetivo Transformar los datos de entrada mediante un conjunto de neuronas que son conectadas localmente desde la capa anterior. Esta capa calculará un producto de puntos entre una región concreta de la capa de entrada y los pesos que la conectan a la capa de salida. Normalmente, este proceso suele mantener las mismas dimensiones espaciales Concretamente, esta CNN extraerá un vector de 128 valores a partir de cada imagen que se le muestre. Evidentemente, aunque el entrenamiento de una CNN profunda solo sea necesario una vez, requiere un enorme número de iteraciones y una base de datos de imágenes lo suficientemente extensa como para poder llegar a generar mediciones de cualquier cara. Por este motivo, la red neuronal dedicada a la extracción de características que se va a usar en este proyecto será una proporcionada por OpenFace que ya viene previamente entrenada con un conjunto de imágenes de rostros para ser capaz de obtener el vector de características y que además, ha sido implementada en la librería Face Recognition. Segundo el autor de la librería, este modelo presenta un porcentaje de precisión del 99.38, por lo que se puede decir que se perderá muy poca información antes de llegar a la última etapa, la comparación y clasificación de estos datos.

2.6. Comparación y clasificación

Para la última fase del sistema de reconocimiento facial será necesario comparar el vector de características que se corresponde con la cara en cuestión con las analizadas previamente para identificar si se trata de una persona conocida, o por el contrario, se trata de una cara desconocida todo utilizando:

Modelo LBPHFace

Los modelos EigenFace y FisherFace toman un enfoque holístico para la identificación fácil. La información es tratada como un vector con un espacio dimensional muy alto. El enfoque realizado por EigenFace maximiza la dispersión total, el cual puede conllevar problemas si la varianza se debe por una fuente externa, ya que los componentes con una varianza máxima sobre todas las clases existentes no son necesariamente útiles para la clasificación. Para preservar alguna información discriminatoria, se aplica el LDA y se optimiza, como se describió para el modelo FisherFace. Ahora, la idea no es mirar la imagen por completo como un vector con una gran dimensionalidad, sino describir solo las características locales de un objeto. Las características que se extraen de esta forma tendrán una baja dimensionalidad implícitamente. En esto se basa el operador de Patrones Binarios Locales (LBP en inglés). Se basa en el análisis de texturas en 2D. La idea principal de LBP es resumir la estructura local de una imagen mediante la comparación de cada píxel con sus píxeles vecinos. Se toma un píxel como centro y se

limita el valor de los vecinos. Es decir, si la intensidad del píxel central es mayor que su vecino, este se denota con un cero, y si, por el contrario, la intensidad del vecino es mayor o igual a la intensidad del vecino central, entonces se denota con un uno.

3. Código

3.1. Código 1: Capturando Rostro

En este código se genera un enfoque del rostro encerrándolo en un cuadrado de color verde, para esto pudimos comprobar que son necesarios algunos factores como la iluminación para que se pueda enfocar bien el rostro, además se generan y guardan imágenes (fotos) las cuales serán ocupadas generar y ampliar nuestra DB.

```

1 import cv2
2 import os
3 import imutils
4
5 personName = 'Esteban'
6 dataPath = 'C:/Users/esteb/OneDrive/Escritorio/Reconocimiento_facial/Data' #Ruta donde estaran los datos
7
8 personPath = dataPath + '/' + personName
9
10 if not os.path.exists(personPath):
11     print('Carpetas creadas: ', personPath)
12     os.makedirs(personPath)
13 #WebCam o Video
14 cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)
15 #cap = cv2.VideoCapture('alvaro.mp4')
16
17 faceClassif = cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default.xml')
18 count = 0
19
20 while True:
21
22     ret, frame = cap.read()
23     if ret == False: break
24     frame = imutils.resize(frame, width=640)
25     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
26     auxFrame = frame.copy()
27
28     faces = faceClassif.detectMultiScale(gray,1.3,5)
29
30     for (x,y,w,h) in faces:
31         cv2.rectangle(frame, (x,y),(x+w,y+h),(0,255,0),2)
32         rostro = auxFrame[y:y+h,x:x+w]
33         rostro = cv2.resize(rostro,(150,150),interpolation=cv2.INTER_CUBIC)
34         cv2.imwrite(personPath + '/rostro_{}.jpg'.format(count),rostro)
35         count = count + 1
36     cv2.imshow('frame',frame)
37
38     k = cv2.waitKey(1)
39     if k == 27 or count >= 400:
40         break
41
42     cap.release()
43     cv2.destroyAllWindows()
44

```

3.2. Código 2: Entrenamiento

En este código lo que hacemos es entrenar nuestro programa por medio del algoritmo LBPH.

Los pasos involucrados para lograrlo son:

- creación de conjuntos de datos
- adquisición de rostros
- extracción de características
- clasificación

```

1 import cv2
2 import os
3 import numpy as np
4
5 dataPath = 'C:/Users/esteb/OneDrive/Escritorio/Reconocimiento_facial/Data' #Ruta donde estan los datos
6 peopleList = os.listdir(dataPath)
7 print('Lista de personas: ', peopleList)
8
9 labels = []
10 facesData = []
11 label = 0
12
13 for nameDir in peopleList:
14     personPath = dataPath + '/' + nameDir
15     print('Leyendo las imágenes')
16
17     for fileName in os.listdir(personPath):
18         print('Rostros: ', nameDir + '/' + fileName)
19         labels.append(label)
20         facesData.append(cv2.imread(personPath+'/'+fileName,0))
21
22     label = label + 1
23
24
25 face_recognizer = cv2.face.LBPHFaceRecognizer_create()
26
27 # Entrenando el reconocedor de rostros
28 print("Entrenando...")
29 face_recognizer.train(facesData, np.array(labels))
30
31 # Almacenando el modelo obtenido
32 face_recognizer.write('modeloLBPHFace.xml')
33 print("Modelo almacenado...")
34

```

3.3. Código 3: Reconocimiento facial

Este código detecta y reconoce el rostro de una persona, si la persona no está en la base de datos el código no reconocerá el rostro pero si lo detectara, todo esto utilizando el método LBPHface.

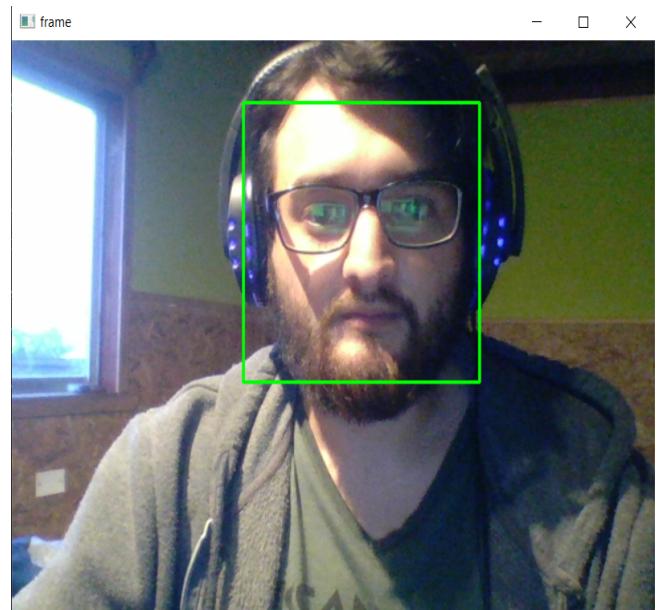
```

1 import cv2
2 import os
3
4 dataPath = 'C:/Users/esteb/OneDrive/Escritorio/Reconocimiento_facial/Data' #Ruta de los datos
5 imagePaths = os.listdir(dataPath)
6 print('imagePaths=',imagePaths)
7
8
9 face_recognizer = cv2.face.LBPHFaceRecognizer_create()
10
11 # Leyendo el modelo
12
13 face_recognizer.read('modeloLBPHFace.xml')
14
15 #Webcam o video
16 cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)
17 #cap = cv2.VideoCapture('alvaro.mp4')
18
19 faceClassif = cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default.xml')
20
21 while True:
22     ret,frame = cap.read()
23     if ret == False: break
24     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
25     auxFrame = gray.copy()
26
27     faces = faceClassif.detectMultiScale(gray,1.3,5)
28
29     for (x,y,w,h) in faces:
30         rostro = auxFrame[y:y+h,x:x+w]
31         rostro = cv2.resize(rostro,(150,150),interpolation= cv2.INTER_CUBIC)
32         result = face_recognizer.predict(rostro)
33
34         cv2.putText(frame,'{}' .format(result),(x,y-5),1,1.3,(255,255,0),1,cv2.LINE_AA)
35
36         # LBPHface
37         if result[1] < 70:
38             cv2.putText(frame,'{}' .format(imagePaths[result[0]]),(x,y-25),2,1.1,(0,255,0),1,cv2.LINE_AA)
39             cv2.rectangle(frame, (x,y),(x+w,y+h),(0,255,0),2)
40         else:
41             cv2.putText(frame,'Desconocido',(x,y-20),2,0.8,(0,0,255),1,cv2.LINE_AA)
42             cv2.rectangle(frame, (x,y),(x+w,y+h),(0,0,255),2)
43
44     cv2.imshow('frame',frame)
45     k = cv2.waitKey(1)
46     if k == 27:
47         break
48
49     cap.release()
50     cv2.destroyAllWindows()
51

```

4. Pruebas:

En el primer código se detecta el rostro de una persona para guardar 300 imágenes con el rostro de la persona en una carpeta que se le asigna un nombre antes de correr el código. La siguiente imagen muestra una prueba que se hizo en una aplicación instalada de forma local, en donde se detecta el rostro.

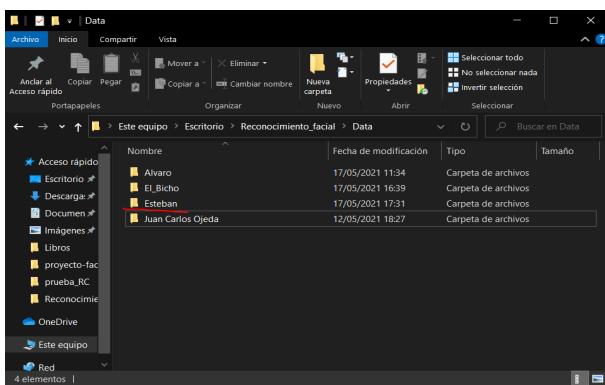


En la siguiente imagen se muestra la carpeta Data en donde se guardarán todas las carpetas con fotos de los rostros que se desean guardar:

Nombre	Fecha de modificación	Tipo	Tamaño
■ Data	17/05/2021 17:30	Carpeta de archivos	
■ alvaro	15/05/2021 19:22	Archivo MP4	10.209 KB
■ capturandoRostros	17/05/2021 17:00	Python File	2 KB
■ cr7_1	17/05/2021 13:50	Archivo MP4	16.225 KB
■ cr7_2	17/05/2021 16:36	Archivo MP4	75.374 KB
■ CR7_entre	17/05/2021 16:06	Archivo MP4	13.187 KB
■ entrenamiento	17/05/2021 16:49	Python File	1 KB
■ michael-scott1	10/05/2021 13:41	Archivo JPG	27 KB
■ modeloLBPHFace	17/05/2021 16:42	Documento XML	132.480 KB
■ reconocimiento_facial	17/05/2021 16:55	Python File	2 KB

En la carpeta Data encontraremos todas las carpetas con los nombres de la persona y sus 300 rostros para poder entrenar nuestra red neuronal, en la siguiente imagen se muestra la carpeta "Esteban" que contiene 300 imágenes:

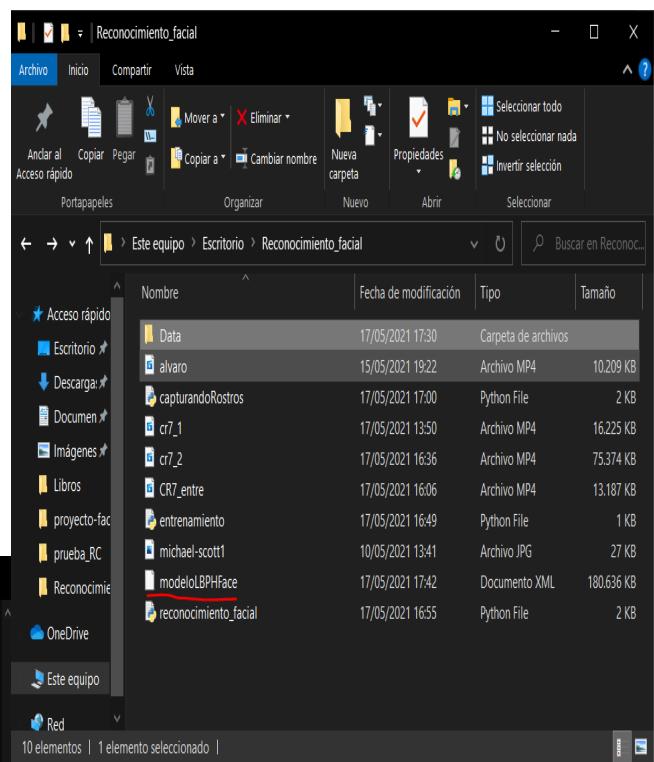
Reconocimiento Facial



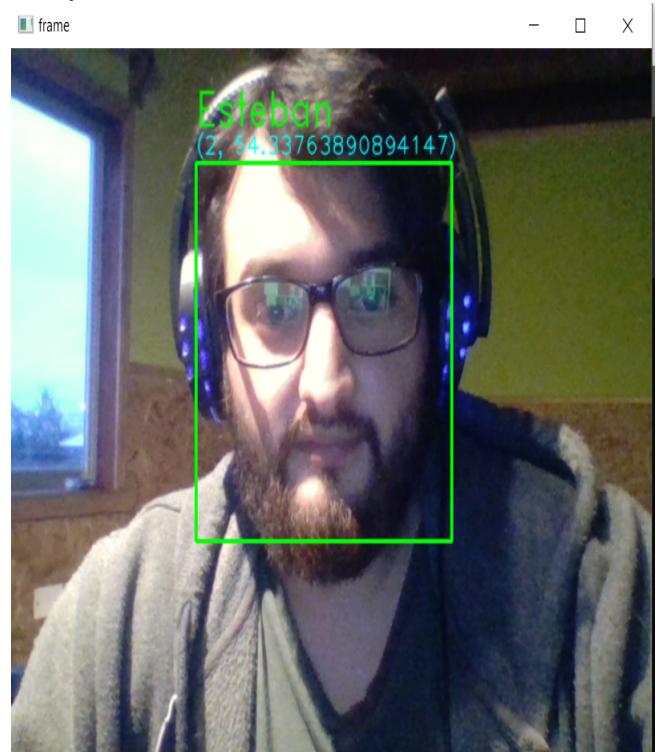
En la siguiente imagen se muestra el entrenamiento de nuestra red neuronal que está contenida en el segundo código. El entrenamiento analiza todas las carpetas y archivos (imágenes) que están contenidas en la carpeta Data.

```
Simbolo del sistema - py_entrenamiento.py
Rostros: Esteban/rostro_173.jpg
Rostros: Esteban/rostro_174.jpg
Rostros: Esteban/rostro_175.jpg
Rostros: Esteban/rostro_176.jpg
Rostros: Esteban/rostro_177.jpg
Rostros: Esteban/rostro_178.jpg
Rostros: Esteban/rostro_179.jpg
Rostros: Esteban/rostro_180.jpg
Rostros: Esteban/rostro_181.jpg
Rostros: Esteban/rostro_182.jpg
Rostros: Esteban/rostro_183.jpg
Rostros: Esteban/rostro_184.jpg
Rostros: Esteban/rostro_185.jpg
Rostros: Esteban/rostro_186.jpg
Rostros: Esteban/rostro_187.jpg
Rostros: Esteban/rostro_188.jpg
Rostros: Esteban/rostro_189.jpg
Rostros: Esteban/rostro_190.jpg
Rostros: Esteban/rostro_191.jpg
Rostros: Esteban/rostro_192.jpg
Rostros: Esteban/rostro_193.jpg
Rostros: Esteban/rostro_194.jpg
Rostros: Esteban/rostro_195.jpg
Rostros: Esteban/rostro_196.jpg
Rostros: Esteban/rostro_197.jpg
Rostros: Esteban/rostro_198.jpg
Rostros: Esteban/rostro_199.jpg
Rostros: Esteban/rostro_200.jpg
Rostros: Esteban/rostro_201.jpg
```

El entrenamiento dejara un modelo que se guardara en la carpeta principal. Este modelo nos servirá para poder reconocer el rostro de una persona en un video o Webcam. En la siguiente imagen se puede ver la carpeta principal con el archivo "modeloLBPHFace" la cual es el modelo que se generó en el entrenamiento:

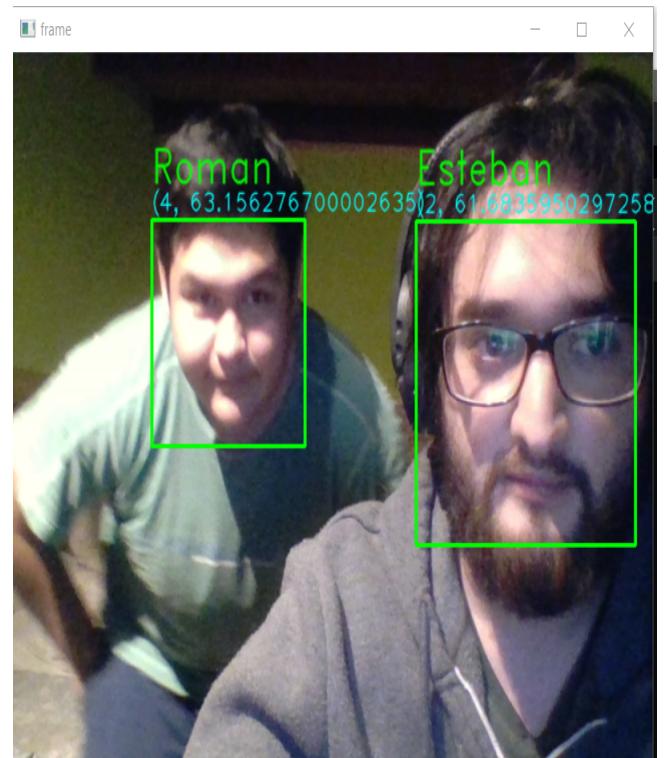
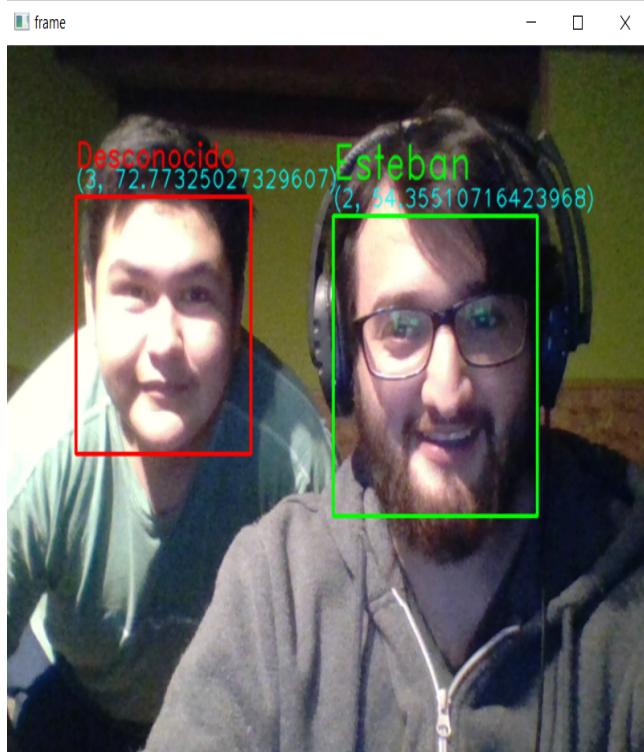


A continuación, se probó el modelo, resultando con éxito como se esperaba. el código 3 puede reconocer el rostro de una persona que haya pasado por los procesos anteriores (captura de rostro y entrenamiento de la red neuronal). En la siguiente imagen se muestra el modelo funcionando y detectando y reconociendo el rostro:



El modelo también se probó con dos personas para verificar

que podía detectar dos rostros. No tuvo problemas en detectar los dos rostros y también pudo reconocer uno (que está en el modelo que se entrenó previamente), el otro rostro aparece como desconocido ya que no está en la base de datos, por lo tanto, el modelo no lo puede identificar y le da una etiqueta de "Desconocido". En la siguiente imagen se muestra el resultado que se obtuvo:



5. Código complementación

A continuación, se mostrará el código que se utilizó para hacer la estimación de punto de referencia, que se ocupó para los ejemplos que se utilizó en el punto 2.3 y 2.4.

Se hizo una prueba más, se agregó el rostro que no identificaba el modelo a la base de datos y se volvió a entrenar. Posteriormente se probó el modelo para ver si podía detectar y reconocer ambos rostros con sus respectivas etiquetas (mencionar que las etiquetas de los nombres se sacan respecto a los nombres de las carpetas de los rostros que fueron guardados en la carpeta "Data"). Como podemos ver en la siguiente imagen, ambos rostros son detectados y reconocidos con su respectiva etiqueta.

```

1 import cv2
2 import dlib
3
4 detector = dlib.get_frontal_face_detector()
5
6 predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
7
8 img = cv2.imread("el_bicho.jpg")
9
10 gray = cv2.cvtColor(src=img, code=cv2.COLOR_BGR2GRAY)
11
12 faces = detector(gray)
13
14 for face in faces:
15     x1 = face.left()
16     y1 = face.top()
17     x2 = face.right()
18     y2 = face.bottom()
19
20     landmarks = predictor(image=gray, box=face)
21     for n in range(0, 68):
22
23         x = landmarks.part(n).x
24         y = landmarks.part(n).y
25
26         cv2.circle(img=img, center=(x, y), radius=3, color=(0, 255, 0), thickness=-1)
27
28 cv2.imshow(winname="Face", mat=img)
29
30 cv2.waitKey(delay=0)
31
32 cv2.destroyAllWindows()
33

```

6. Conclusión

Uno de los problemas que tuvimos a realizar este trabajo fue que los códigos en los cuales nos guiamos al comienzo tenían una gran cantidad de fallas algunas de las cuales no pudimos encontrar la solución por lo cual se decidió apoyarnos en otros códigos para así poder llegar a un programa funcional de reconocimiento facial.

Para poder entender mejor el funcionamiento se implementaron cinco códigos, tres de los cuales colaboran entre sí generando el programa principal, los otros dos nos ayudan a entender mejor el funcionamiento del programa permitiéndonos visualizar pasos que se producen dentro de este pero no nos son mostrados de forma gráfica.

También se pudo comprobar que el código de entrenamiento requería 300 imágenes dentro de la DB para poder reconocer cuando en un mismo video o transmisión existen más de un rostro.

A pesar de todos los inconvenientes se pudo implementar y lograr el objetivo de un programa de detección facial.

Mencionar también que para las pruebas era necesario tener el lugar iluminado para que se pueda detectar el rostro. Además, en la captura de rostro se necesitaba que la persona haga algún tipo de gesto para que el modelo que se entrena pueda tener varias referencias de los rostros.