

Utilización de una Red Neuronal para la Detección de Personas en una Raspberry Pi

**Autor:**

Llanes, Federico.

Índice

1. Objetivos………………………………………………………………………………………………..…5

2. Requerimientos…………………………………………………………………………………………5

3. Materiales……………………………………………….…..……………………………………………6

4. Librerías….……………………………………………….…..……………………………………………6

5. Algoritmos……………………………………………….…..……………………………………………7

6. Instalación……………………………………………………………………………………………….12

7. Programa…………………………………………………………………………………………………15

7.1 Estructura……………………………………………………………………………….……15

7.2 Archivos y carpetas soportes……………………….……………………………….15

7.3 Programas …………………………………………………………..………………………16

8. Pruebas……………………………………………………………………………………………………26

9. Conclusión…………………………………………………………………………………………….…31

**Objetivos:**

El objetivo de este proyecto es poder desarrollar, utilizando redes neuronales y librerías abiertas de Python, un programa de reconocimiento facial. Se espera que el programa nos permita reconocer usuarios en nuestra base de datos utilizando la cámara, y también agregar y quitar usuario de nuestras bases de datos, mantener un registro de cada detección que realiza, un menú, y la posibilidad de cambiar los parámetros.

**Requerimientos:**

Requerimientos funcionales:

* El sistema debe poder detectar objetos las caras con una alta efectividad. (Debería ver el objeto y reconocerlo)
* La confusión entre los distintos usuarios de la base de datos debe ser baja
* Se debe utilizar las herramientas de una manera que sea fácil de modificar para proyectos futuros

Requerimientos no funcionales:

* El software no debe calentar la placa, ni perjudicar su funcionamiento.
* El FrameRate debe ser aceptable para posibles usos prácticos.

**Materiales:**

* Una Raspberry PI 3 o superior.
* Un módulo de cámara.
* Un teclado y un mouse.
* Un monitor HDMI
* Un cable HDMI
* Una Computadora

Este tutorial usa del monitor y del mouse para una instalación sencilla, también se puede conectar a la interfaz gráfica de la RaspBerry utilizando el puerto ETHERNET conectado a nuestra computadora, aunque eso suele generar conflictos en la conectividad WiFi de la misma, y no nos permite conectar nuestras RaspBerry a una conexión de internet cableada.

**Librerías:**

El código que utilizaremos para realizar la detección está escrito en lenguaje Python3. Para eso utilizaremos varias librerías de Python.

***OpenCV*:**

OpenCV (Open Source Computer Vision) es una librería open source relacionada al tratamiento de imágenes e inteligencia artificial. Contiene una gran cantidad de algoritmos útiles que permiten identificar rostros, reconocer objetos, clasificar objetos, detectar movimientos con las manos, entre otras cosas.

Se utilizará para guardas las imágenes en el log, y modificar la imagen que muestra nuestro programa (agregando texto, etc). Pero su función principal es detectar las caras que hay en la foto.

***Dlib:***

Dlib es una librería que contiene algoritmos de “machine learning” y herramientas para resolver problemas del mundo real. Es muy utilizada en el mundo académico, y también en la industria, es de open source y se puede utilizar para cualquier aplicación sin costo alguno.

Se utilizará para comparar las caras detectadas con OpenCV y comprararlas con aquellas en nuestra base de datos.

**Imutils:**

Una librería que permite realizar algunas funciones básicas relacionadas con el procesamiento de imágenes, como rotar la imagen, cambiar su resolución, etc.

Se utilizará para cambiar la resolución de la imagen proveniente de la cámara.

**Pickle:**

Los archivos pickle representan un objeto de Python como una cadena de Bytes. Se usan generalmente para almacenar resultados de un programa que pueda ser utilizado por otro programa.

Se utilizará para guardar la información relacionada a cada usuario, además de los parámetros de funcionamiento.

**Otras librerías a utilizar**

**OS**: Nos permite realizar funciones a nivel sistema operativo, como correr comandos, crear archivos, etc. Se utilizará para correr programas desde otro programa, renombrar archivos, y crear carpetas.

**Shutil**: Permite operar con archivos. Se utilizará para borrar carpetas.

**Time**: Se usa para operaciones relacionadas con tiempo. Se utilizará para poner el programa a dormir luego de reconocer a un usuario.

**Datetime**: Sirve para manipular fechas. En nuestro caso lo utilizaremos para guardar información referida a la hora en la cual se realizó el reconocimiento.

**RPi.GPIO**: Sirve para controlar los canales GPIO de la Raspberry. Se utilizará para prender un láser al reconocer a un usuario.

**Decimal**: Sirve para operaciones aritméticas con decimales. Se utilizará para eliminar cifras significativas.

**Numpy**: Una biblioteca para funciones matemáticas de alto nivel, especialmente útil para tratar con vectores y matrices. Se utilizará para tratar imágenes como vectores, y poder medir su intensidad lumínica.

**Face\_recognition**: Una librería basada en dlib para facilitar el reconocimiento de imágenes.

**Algorítmos:**

Hay 3 algorítmos principales a tener en cuenta que implementamos en nuestro programa. Un clasificador cascada para reconocimiento de caras(haarcascade\_frontalface\_default) , un reconocedor facial(face\_encodings), y un comparador(compare\_faces)

**El clasificador cascada “haarcascade\_frontalface\_default” (de openCV):**

El algoritmo “Haar Cascade” para la detección de objeto, fue propuesto en el 2001 por Paul Viola y Michael Jones. Es básicamente una función cascada (Se mencionará más tarde el significado de esto) entrenada con positivos (donde está presente el objeto a ser detectado) y negativos (donde el objeto a ser detectado no está presente). Existen para detección de muchas cosas, como ser ojos, boca, nariz, etc. Nosotros utilizaremos el detector de caras.

El objetivo es extraer características de las imágenes usando “filtros”, estos filtros se llaman “Características Haar” y son los vistos en la Figura siguiente.

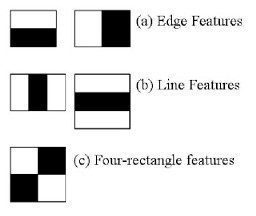


Fig 1 Características Haar

Se pasan todos los filtros en cada parte de la imagen, y por cada iteración, sumamos todas las intensidades de pixeles de la parte negra y la parte blanca. La diferencia del valor entre ambas es el valor de la característica estudiada (Mientrás más alto, más posible que se trata del elemento estudiado)

Por ejemplo, para el análisis de borde en la siguiente imagen.

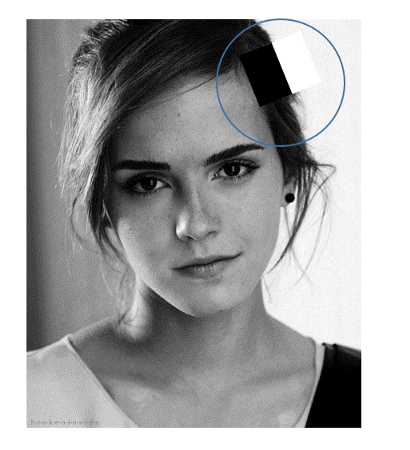


Ilustración 1 Ejemplo del funcionamiento de un filtro Haar

Se pasa el filtro en esa sección del cabello. La imagen tratada está en blanco y negro (algo que siempre se hace cuando tratamos con filtros Haar). Si sumamos la intensidad de pixel (0 a 255, siendo 0 blanco y 255 negro, con los demás valores representando gamas de grises) de la parte Negra del filtro, que justo toca los cabellos, tendremos un resultado muy alto (ya que el pelo aparece oscuro). Sin embargo, la parte blanca, que da con el fondo de la imagen, recibirá un valor muy bajo ya que el fondo se ve blanco. El valor de la resta de estos valores será entonces un número alto, y podemos asumir, que dimos con un borde. En la ilustración siguiente, podemos notar, entonces, que el reconocimiento devuelve un resultado muy similar al original, y como ya dijimos, asumimos que dimos con un borde.

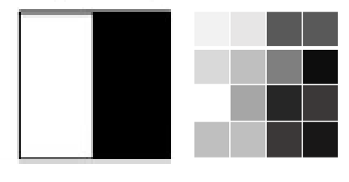


Ilustración 2 Filtro utilizado y resultado Obtenido.

Se hace esto para toda la imagen (en nuestro caso la cara), para obtener todas las características que tendrá nuestra imagen, como se ve en la Fig. 2 En un caso real, tomamos las 6000 características más relevantes de todas las calculadas, solo con 200 podemos tener una precisión del 95%

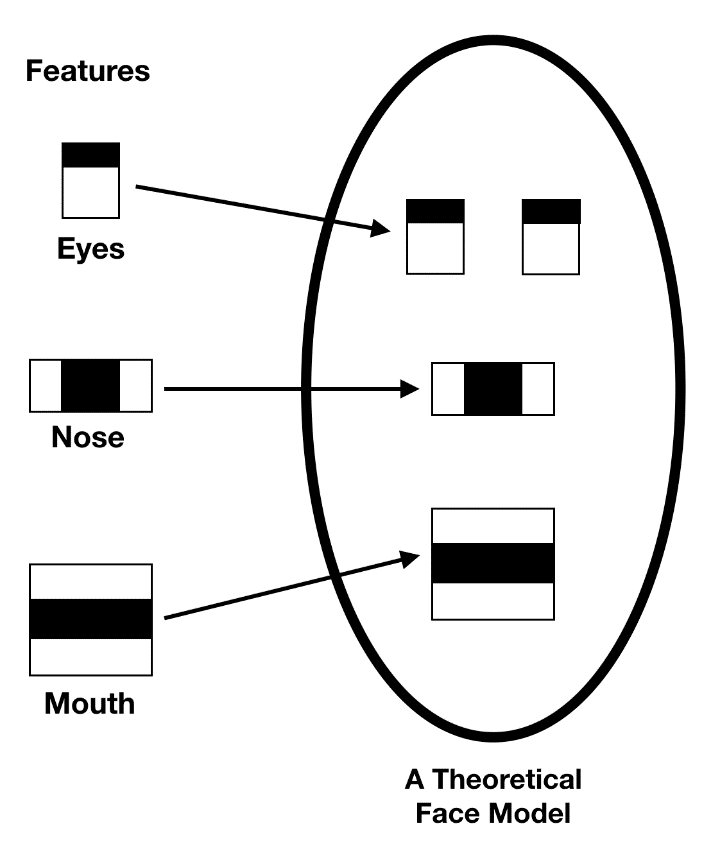


Fig 2 Un modelo teórico sobre simplificado de una cara.

La parte de cascada se refiere al método para hacer el análisis. En vez de aplicar las 6000 características a la imagen, las características se dividen en distintas etapas, y se aplican una por una. Si la imagen no pasa la primera etapa, se descarta, y no se continúa el entrenamiento. En el algoritmo utilizado, hay 38 etapas, y las primeras 5 etapas tienen 1, 10, 25, 25 y 50 características cada una.

**face\_encodings:**

face\_encondigs es una de las tantas subfunciones que nos brinda la librería face\_recognition basada en dlib.   
  
Esta devuelve el resultado de la función de dlib llamada “compute\_face\_descriptor” en un array.

Compute\_face\_descriptor a su vez, agarra el rectángulo donde se encuentra la cara de la persona, y nos devuelve un vector de 128D. Mientras más similares los números de dos arrays sean, más probables es que se trata de la misma persona.

El algoritmo que se entrenó para generar el vector que mejor “represente” a cada cara, fue entrenado con una red convolucional.

La red utilizada por dlib no es pública, por lo tanto, no sabemos cuántos parámetros tiene o cuales son sus capas. Sin embargo, podemos analizar el funcionamiento de la CNN de FaceNet en la Figura siguiente, que realiza la misma función

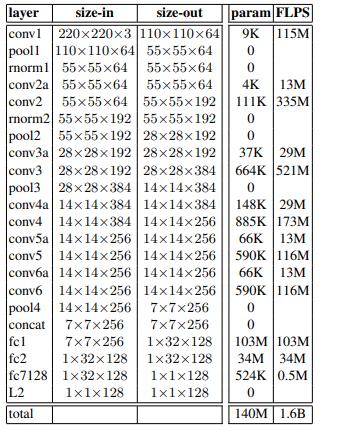


Fig 3 La CNN utilizada por FaceNet para entrenar el proceso de generar el vector "descriptor"

La red se entrena para que la distancia euclidiana entre dos pares de imágenes correspondiente a la misma persona sea mínima, mientras que la distancia entre dos pares de imágenes correspondientes a distintas personas sea máxima.

**compare\_faces:**

compare\_face es una de las tantas subfunciones que nos brinda la librería face\_recognition basada en dlib.

Lo que hace es medir la distancia euclidiana entre el array de 128 elementos de la cara que está siendo procesada, con la cara ya procesada y guardada en nuestra base de datos.

La distancia euclidiana está determinada por:



Si la distancia es menor a 0.6, se considera que se trata de la misma persona. Este número es obtenido mediante la experiencia y se considera el mejor “trade-off” entre el riesgo de no reconocer a una persona y confundirla con otra

**Instalación:**

La instalación dura varias horas, en algunas partes tendremos que esperar más de 4 o 5 horas mientras la Raspberry procesa los datos. Se recomienda hacer esto con tranquilidad, y con otras actividades en mente, ya que por cada línea de comando tendremos entre 10 minutos y algunas horas de espera. Considerando que se requiere descargar varios GBs en librerías, es recomendable tener una buena conexión a internet.

Ejecutamos en la terminal de comandos.

* sudo raspi-config

Seleccionamos “7 Advanced options” y luego “A1 Expand Filesystem”. Cuando haya terminado, volvemos a la consola con la tecla escape y reiniciamos la Raspberry

* sudo reboot

Actualizamos el Sistema.

* sudo apt-get update && sudo apt-get upgrade

Instalamos algunas herramientas para desarrolladores

* sudo apt-get install build-essential cmake unzip pkg-config

Instalamos librerías de imagenes y videos, importantes para el tratamiento de imágenes.

* sudo apt-get install libjpeg-dev libpng-dev libtiff-dev
* sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
* sudo apt-get install libxvidcore-dev libx264-dev

Instalamos GTK.

* sudo apt-get install libgtk-3-dev
* sudo apt-get install libcanberra-gtk\*

Instalamos paquetes con “optimizaciones numéricas para OpenCV”.

* sudo apt-get install libatlas-base-dev gfortran

Instalamos la cabecera de desarrollador de python3.

* sudo apt-get install python3-dev

Instalamos el pip

* wget <https://bootstrap.pypa.io/get-pip.py>
* sudo python3 get-pip.py

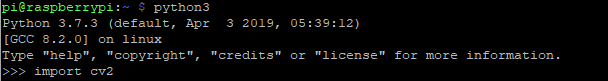
Instalamos el numpy que es un prerequisito para el openCV

* sudo pip3 install numpy

Y finalmente instalamos el OpenCV

* sudo apt-get install python3-opencv

Si hicimos todo bien hasta aquí, el openCV debería estar instalado. Una pequeña prueba es entrar a python3 e importar opencv.



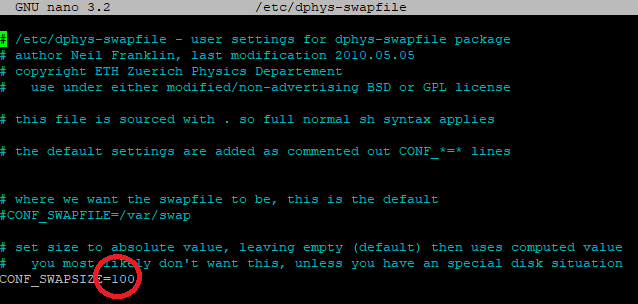
Si no hay mensaje de error, entonces hicimos todo bien.

Ahora instalamos el dlib, esta es la parte más complicada de la instalación, debido al tamaño de la librería.

Empezamos aumentando el tamaño del swap file.

* sudo nano /etc/dphys-swapfile

Luego, cambiamos el valor de 100 a 1024, guardamos y salimos



Actualizamos el swap file size.

* sudo /etc/init.d/dphys-swapfile stop
* sudo /etc/init.d/dphys-swapfile start

Ahora cambiamos las opciones de boot para evitar que la interfaz gráfica de raspberry funcione. Hacemos esto porque la misma ocupa muchos recursos, y necesitamos toda la capacidad de procesamiento de nuestra raspberry. Haciendo esto, nuestra raspberry va a bootear en modo terminal.

* sudo raspi-config

Boot Options => Desktop / CLI => Console Autologin

Luego salimos y volvemos a entrar a la configuración

* sudo raspi-config

Advanced Options => Memory Split

Esto se hace para reducer la memoría que la RaspBerry le da al GPU.

Luego, reiniciamos nuestra raspberry pi.

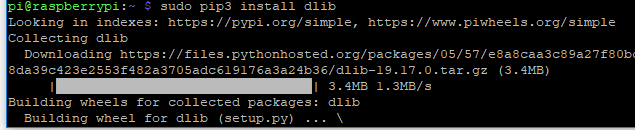
* sudo reboot

Instalamos algunos prerequisitos

* sudo apt-get install libboost-all-dev
* sudo pip3 install scipy
* sudo pip3 install scikit-image

Finalmente instalamos dlib. Este proceso puede demorar varias horas.

* sudo pip3 install dlib



No se asusten al ver esto. Quedará así por un largo tiempo.

Cuando se termine de instalar, volvemos a poner el swap size en 1000, la memory Split en 64, y para que inicie con la interfaz gráfica.

Instalamos algunas utilidades extras :

* sudo pip3 install face\_recognition
* sudo pip3 install imutils

¡Ahora bajamos el programa realizado en este proyecto!

* git clone <https://github.com/ushuaia213/reconocimiento.git>

Finalmente nos aseguramos que la cámara esté activada. En la interfaz gráfica vamos al botón de inicio, preferences y luego Raspberry Pi Configuration.

En el menú de interfaces nos aseguramos que la cámara está en modo “enabled”

Para inciar el programa basta con

* cd reconocimiento
* python3 integrado.py

**Programa:**

**Estructura:**

En la Fig siguiente se puede observar el árbol con los archivos correspondientes al programa

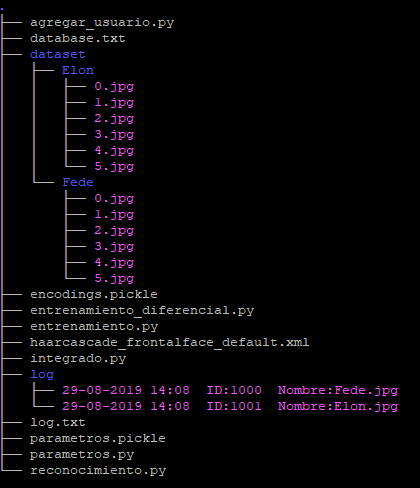


Fig 4 Árbol de archivos de la carpeta contenedora del programa.

**Carpetas y archivos soporte del Programa:**

Carpeta dataset: En esta carpeta, van las carpetas donde están las fotos correspondientes a cada usuario. Para cada Usuario las fotos deben ser guardadas con el nombre del usuario.

Carpeta log: Cada vez que se realice un reconocimiento con éxito, guardaremos la imagen tomada en el momento del mismo.

log.txt: Cada vez que se realice un reconocimiento, se escribirá aquí, a la persona que reconoció, junto son su identificador y la hora del reconocimiento.

Ejemplo:

ID: 1002 Nombre: Jose Horario: 29-08-2019 19:37

ID: 1000 Nombre: Fede Horario: 29-08-2019 21:08

haarcascade\_frontalface\_default.xml: Los valores utilizados para la detección de rostros explicados con anterioridad.

encodings.pickle: Toda la información relacionada con cada usuario. Su nombre, su ID, y su correspondiente array de *encoding* (notese que hay un array de encoding por cada foto del usuario).

parametros.pickle: Información relacionada a los parámetros de los distintos programas, como por ejemplo, en que pin está conectado el láser(de querer prender uno cuando haya detectado a un usuario), cuantas fotos sacar para agregar nuevos usuarios a la base de datos, etc.

Database.txt: Aquí se guarda en un archivo de texto todos los usuarios que están registrados en la red, y cuales son sus correspondientes IDs. Por ejemplo,

ID: 1000 Nombre: Fede

ID: 1001 Nombre: Elon

ID: 1002 Nombre: Jose

**Programas:**

Hay 6 programas diferentes en la carpeta del reconocimiento. Cada uno de ellos cumple una función, agregar usuarios, entrenar la red, re-entrenar la red (luego de agregar nuevos usuarios), empezar el reconocimiento, cambiar los parámetros del programa, y un menú para poder ingresar con más facilidad a todas estas opciones y otras dedicadas.

**Agregar\_usuario.py**

Este programa nos permite agregar un nuevo usuario a nuestra base de datos. Genera una carpeta con el nombre del usuario en la carpeta database, y luego utilizando la cámara saca el número de fotos indicado por los parámetros y los guarda allí.  
  
Sin embargo, no guarda cualquier foto. Tiene 3 detectores de “problemas”. No guardará fotos que no contengan caras, que tengan poca intensidad de luz o tengan mucho *blurry.*

El programa recoge la imagen proveniente de la cámara, las convierte en gris, y aplica el detector de caras ya visto anteriormente.  
  
Para cada cara detectada, el programa analiza el *blurry* y la intensidad de luz. Para detectar el blurry se utiliza el operador Laplaciano integrado a opencv, que convoluciona la imagen por el kernel laplaciano . Esto es similar a calcular la segunda derivada de una imagen, por ende, remarca los lugares donde hay cambios bruscos en la imagen. Luego tomamos la varianza del resultado. Si el valor de la varianza es menor que un valor fijo, se considera a la imagen borrosa, de no serlo, se la considera “no borrosa”. El valor utilizado para este programa es experimental y fue tomado luego de experimentar con varios valores hasta conseguir uno ideal.

Para la intensidad de luz, se convierte la imagen RGB a HSV, que significa (Hue, Saturation, Value – Matiz, Saturación, Valor). Los primeros dos no nos interesan, pero Value representa la intensidad de luz. Por lo tanto, utilizaremos esta dimensión para promediar todos los pixeles dentro de la cara y medir su intensidad de luz.

Analizando la figura de abajo.

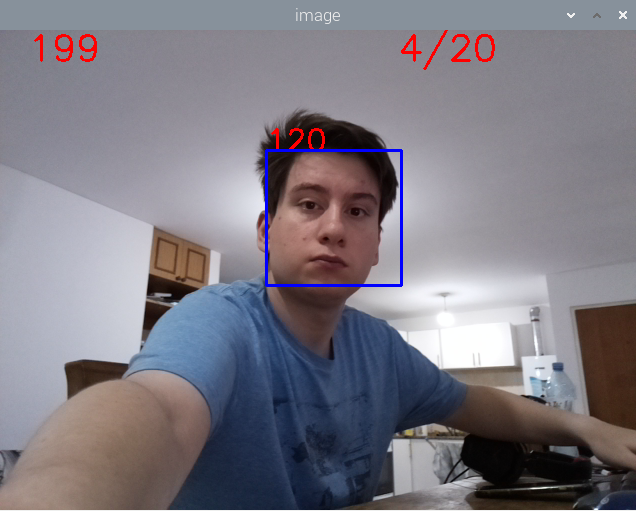


Fig 5 Usuario en proceso de ser agregado a la red.

Hay 3 números a analizar. El de arriba a la izquierda es la intensidad de blurry, si es menor a 100, consideramos que es una imagen borrosa, en esta caso, el número de 199 indica que es una imagen muy clara.  
El número que está junto con la cara es la intensidad de luz, debe ser mayor a 70, en este caso es 120, lo que significa que la imagen está muy bien iluminada.  
Y finalmente arriba a la derecha se puede ver cuantas fotos ya va guardando el programa sobre el número de fotos que debe guardar(este último número editable mediante integrado.py o parámetros.py). Abajo se muestran dos imágenes que serían rechazadas por nuestro programa.

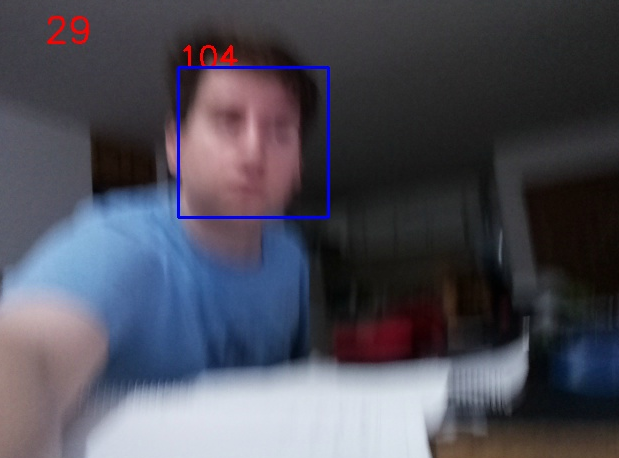
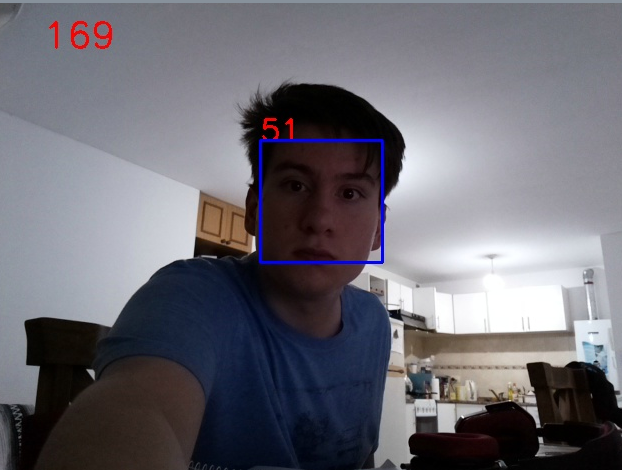


Fig 6 Imagen oscura Fig 7 Imagen borrosa

En la figura siguiente se puede ver como funciona el diagrama de flujo del programa

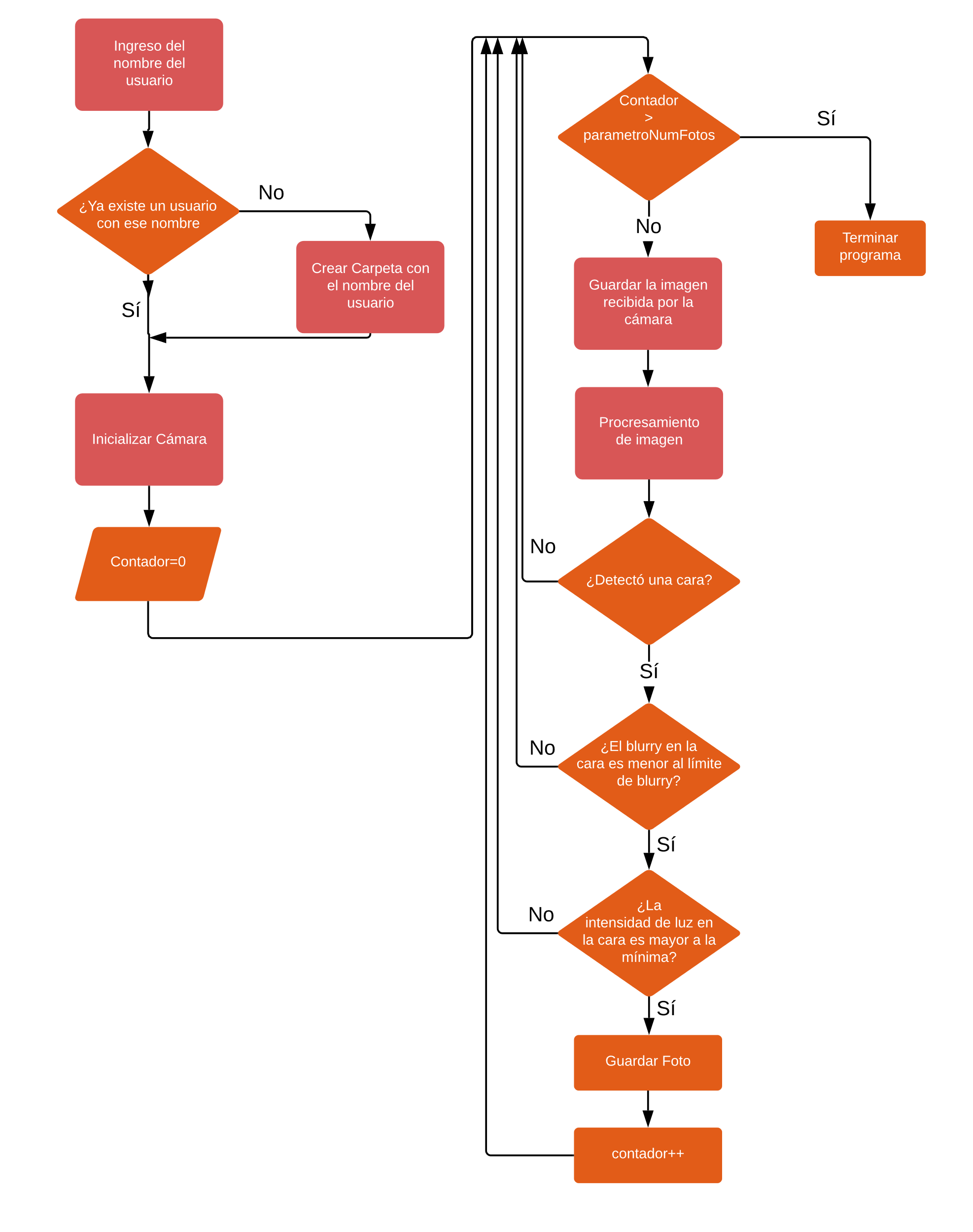


Fig 8 Diagrama de Flujo - agregar\_usuarios.py

**Entrenamiento.py:**

Como se puede inferir por el nombre, este programa se utiliza para entrenar la red. Como hemos estudiado previamente, el proceso de entrenar es simplemente el de generar el vector de codificación de 128D de cada foto de nuestra base de datos. Y junto a él, se guardará también el nombre y el ID del usuario al que le corresponde tal encoding en nuestro fichero encodings.pickle.

El programa va a loopear todas las imágenes existentes en la carpeta database, por cada imagen vamos a recuperar el nombre de la carpeta que la contiene, y se lo otorgaremos al nombre de usuario. Si el usuario es nuevo, le damos un ID y agregamos su información al archivo database.txt, de no ser nuevo, significa que este paso ya se realizó. Luego, generamos el vector encoding y lo guardamos en el fichero pickle junto a su nombre y a su ID. Nótese que si hay 30 fotos de un usuario, tendremos 30 encodings distintos, los cuales guardaremos todos (su id y nombre, por supuesto, serán los mismos para cada uno de los encodings)

El programa también tiene algunas funciones extras, como, por ejemplo, eliminar a las carpetas de usuarios cuyas fotos no haya procesado correctamente (no había ninguna cara), y reorganizar los nombres de los archivos

Este funcionamiento se puede ver en el diagrama de flujo siguiente.

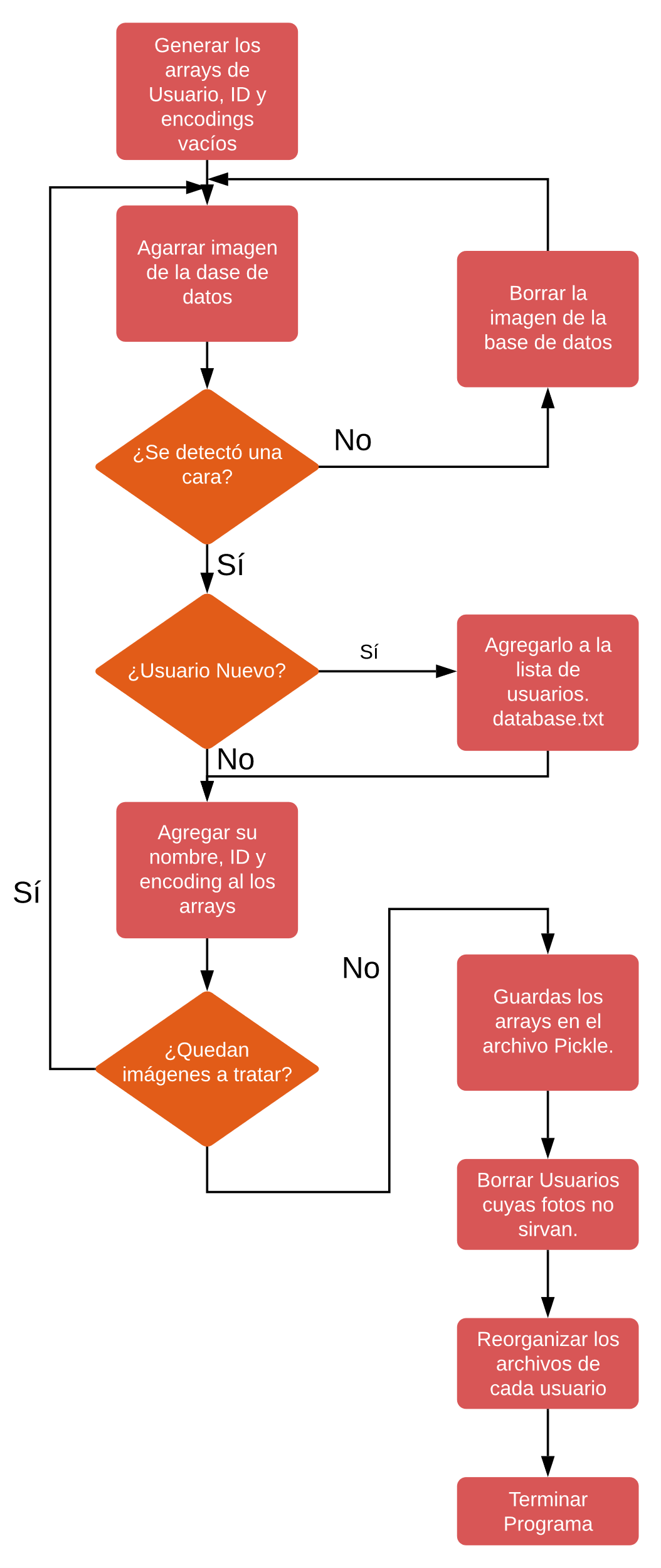


Fig 9 Diagrama de Flujo - entrenar.py

**Entrenamiento\_diferencial.py:**

Este programa se usa únicamente cuando tenemos una base de datos existente y queremos agregar nuevas fotos de un usuario existente, o agregar un nuevo usuario.  
  
Funciona de la misma manera que el programa anterior con algunas excepciones, primero copia en los arrays la información del archivo pickle con Nombre, IDs y el encoding. Luego, cuando analiza cada fichero, se fija si ese fichero ya fue tratado anteriormente, si fue tratado lo ignora, y si no lo fue, significa que es uno de nuestras fotos nuevas y las tratas de la misma manera que antes.

**Parametros.py**

Como se explicó anteriormente, los parámetros relacionados al funcionamiento del programa están en el archivo parámetros.pickle. Este programa lo que hace es, simplemente, modificar los valores de los mismos utilizando un menú de texto y el input del usuario, como se puede observar en la Fig siguiente.

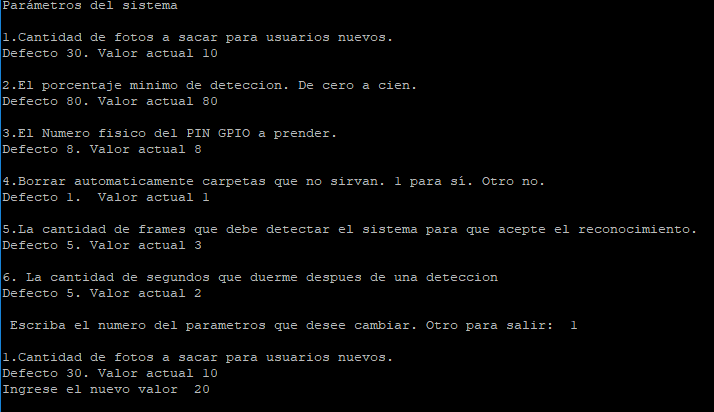
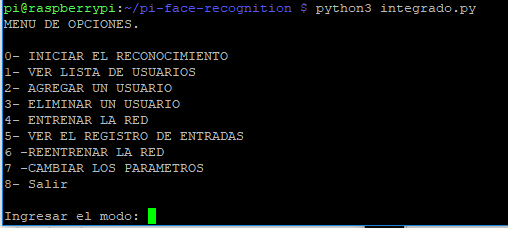


Fig 10 Programa parametros.py, cambio de la cantidad de fotos a sacar para usuarios nuevos.

**Integrado.py**

Este es el archivo que siempre vamos a ejecutar para utilizar nuestro programa. Con un menú de texto, es el lazo que une a todos los programas de la carpeta. Con él, podemos acceder a todas las funcionalidades de nuestro software.

Tiene además algunas extras, como la posibilidad de eliminar usuarios y acceder a los archivos de textos.



**Reconocimiento.py**

Es el programa más importante de todos en la carpeta, y es el que se encarga de hacer el reconocimiento en tiempo real utilizando la cámara, ejemplo en la figura de abajo.

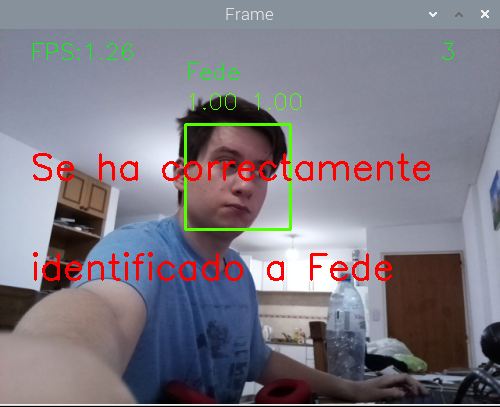


Fig 11 Ejemplo de un reconocimiento

Para empezar, el reconocimiento de caras que realiza internamente es sencillo.

Agarra la imagen proveniente de la cámara, realiza el encoding, y compara las distancias eucledianas con cada uno de los encodings existentes en el archivo pickle, y si la distancia es menor a 0.6, se considera que se trata de la misma persona.

El programa entonces compara con cada foto, si existe un match o no. Y luego, hace un recuento de cuantos matches existen por cada usuario.

Ejemplo. Hay dos personas algo parecidas, cada una con 10 fotos en la base de datos. Se agarra la imagen proveniente de la cámara, y se realiza el encoding. Luego se compara si la distancia euclidiana es menor a 0.6 con cada una de las imágenes de la base de datos (20 en este caso). Esto nos devolverá un array con True para los índices donde hubo un match, y False para los índices donde no los hubo.   
Luego vemos a quien pertenece cada índice donde se indica True, y contamos cuando matches tuvo cada persona. Si hubo 9 matches para Usuario 1 y 2 para el Usuario 2, como el usuario 1 tuvo una cantidad de emparejamientos mayor, entonces consideramos que se trata de esa persona.

Si bien este es el funcionamiento básico del reconocedor de imágenes, en el programa se agregaron varias funciones para agregarle robustez.

Por ejemplo, en la Fig 9 se muestran dos valores en verde 1.00 y 1.00.

El de la derecha representa la confianza de que sea ese usuario. Por ejemplo, si en nuestra base de datos el usuario X tiene 10 fotos y 8 las acepta como similares, pero a las otras 2 no. El resultado será 0.8  
  
El de la izquierda nos da la confianza de que sí es un usuario de la base de datos, sea ese y no otro. Siguiendo el ejemplo anterior, si del otro usuario detectó 1 foto, entonces el resultado será 8/9=0.89, ya que es la cantidad de fotos aceptadas del usuario sobre la cantidad de fotos aceptadas en total.

Ambos parámetros deben ser mayores a un mínimo de detección para que el reconocimiento se acepte (por lo general 0.7).

Aquí se pueden generar dos conflictos. Uno es tener fotos de dos personas distintas en la misma carpeta, en este caso el segundo parámetro daría 1.00 ya que todos los reconocimientos fueron hechos para ese usuario, pero el segundo parámetro daría un número menor, por que solo se reconocieron una porción de las fotos totales del usuario. El otro conflicto, mucho más probable, es generar dos carpetas para un mismo usuario con dos nombres distintos, por ejemplo, Fede y Federico, cada una con 20 imagenes. Ambas detecciones individuales darán un número alto, en referencia al primer parámetro, pero el segundo parámetro será un número bajo ya que el número total de detecciones para cualquiera de ellos, divido las detecciones totales será bajo.

El programa detecta automáticamente este caso, y escribe un mensaje en consola, como se puede apreciar debajo.



Los colores de ambos números juntos con la caja en la cara, cambian dependiendo del nivel de seguridad de que sea el usuario. Si está 100% seguro será verde puro, si está al mismo nivel que el mínimo de detección será amarrillo. Cualquier valor en el medio, variara de manera gradual del amarrillo al verde a medida que aumente el valor de confianza.

Otro de los números que se aprecia en la imagen es FPS, este es simplemente la cantidad de imágenes que el programa procesa en un segundo, por lo general es 1.3 cuando está detectando.

Arriba a la derecha está el número de detecciones consecutivas realizadas. Como una segunda medida de seguridad, para tomar a un reconocimiento como final, debe reconocer a la misma persona por una cantidad de frames determinada (también modificable por comando, generalmente 5). En este caso eran 3 los necesarios para identificar.

Finalmente, al realizar una detección nos dará el mensaje de que se ha detectado al usuario y guardará la foto en la carpeta log además de escribir el reconocimiento en el archivo log.txt

La detección también puede prender un láser u otro dispositivo alimentado por 5V si así se desea, por unos segundos. Durante esos segundos, el programa dejará de reconocer.

El diagrama de bloques se muestra en la figura siguiente.

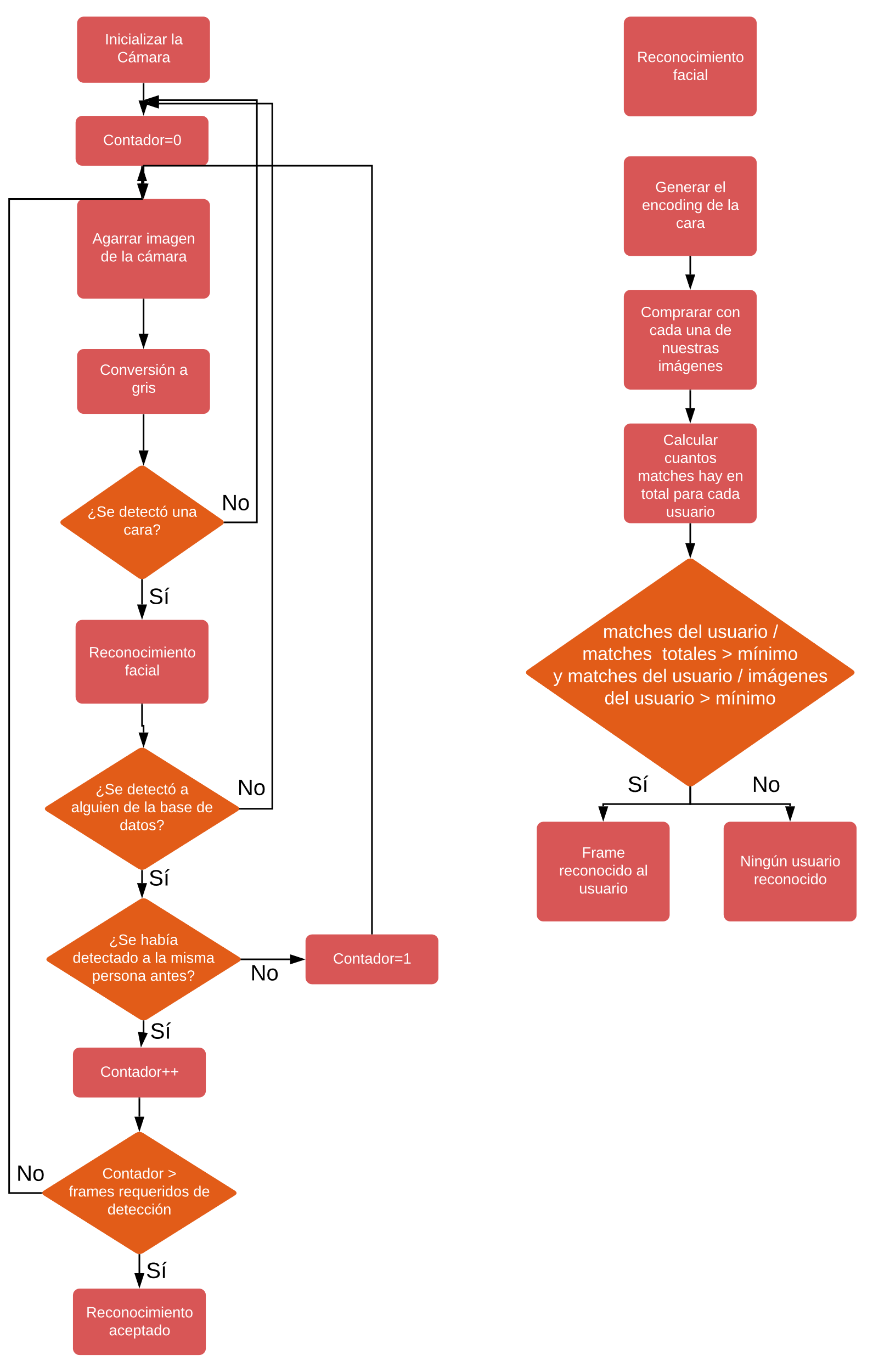


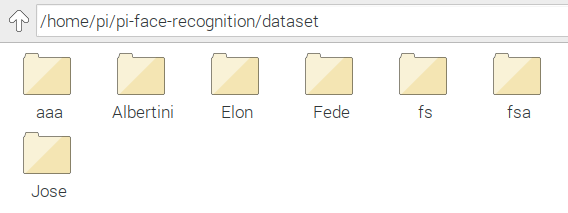
Fig 12 Izquierda, diagrama de bloques del reconocedor facial. Derecha, diagrama de bloques del bloque "Reconocimiento facial" de la figura de la izquierda

**Pruebas:**

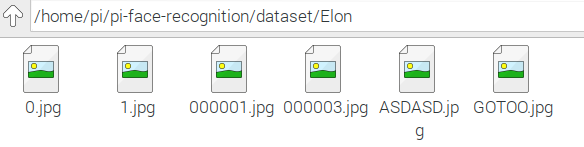
Algunas de las pruebas realizadas incluyen:

* Tener carpetas con usuarios cuyas fotos no sirvan (no hay caras)
* Generar conflicto teniendo más de una carpeta con el mismo usuario.
* Tener carpetas con fotos útiles e inútiles.
* Tener nombres de ficheros desorganizados, para ver si el programa es capaz de acomodarlos.
* Agregar nuevos usuarios para ver si se actualiza correctamente el archivo database.txt

Vamos a probar los primeros cuatros elementos al mismo tiempo.

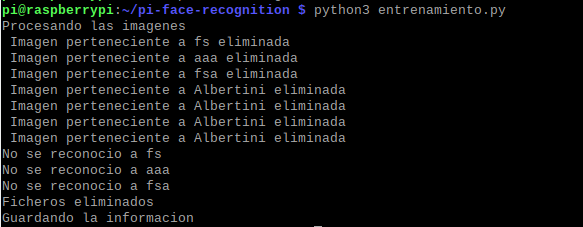


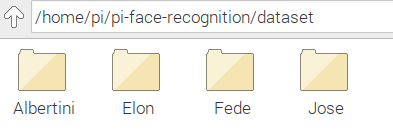
Las carpetas con nombres erráticos no contienen fotos con caras. La Carpeta Albertini tiene 4 fotos que no sirven, además de representar al mismo usuario que Fede. Además de eso, la carpeta con las fotos de Elon tienen nombres desorganizados.

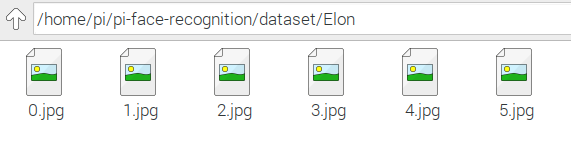


Al correr el programa de entrenamiento, debería borrar esas 3 carpetas, borrar las imágenes que no sirven de Albertini, y reorganizar los nombres de las imágenes de Elon.

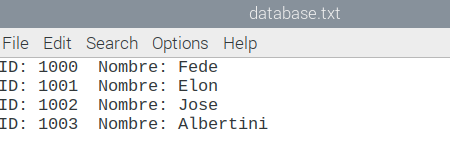
Resultados al correr el programa: El programa nos informa que se han borrado las fotos correspondientes a Albertini y que hubo 3 usuarios que no fueron reconocidos y eliminados. Además de reorganizar los nombres de las fotos de Elon.





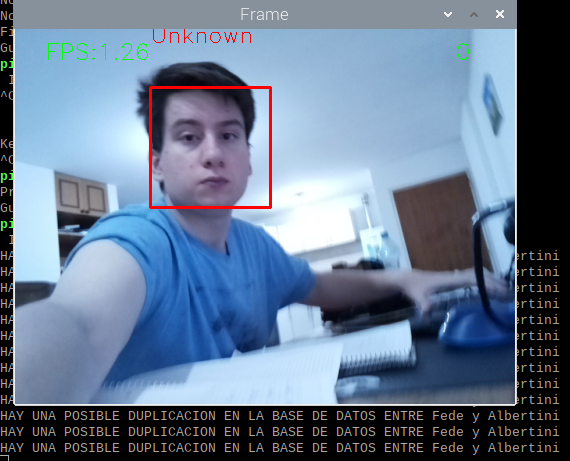


Abriendo el archivo database.txt podemos observar:

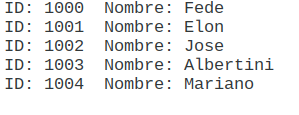


Si iniciamos el reconocimiento entonces:

Detecta lo duplicación como se puede ver en la consola debajo de la imagen



Si agregamos un usuario y luego utilizamos el entrenamiento diferencial. El usuario agregado irá último en la lista, y el resto de la lista quedará igual que antes.



Reconocimiento de dos personas similares.

Las actrices Lili Reinhart y Sarah Gadon(Fig de abajo) son consideradas como “muy parecidas”. Cargando 5 fotos de cada una de ellos en nuestra base de dato, y probando con otras imágenes para saber si el programa puede diferenciarlas correctamente



Fig 13 Actrices Lili Reinhart y Sarah Gadon

Los resultados fueron positivos, no hubo ninguna instancia de confusión.

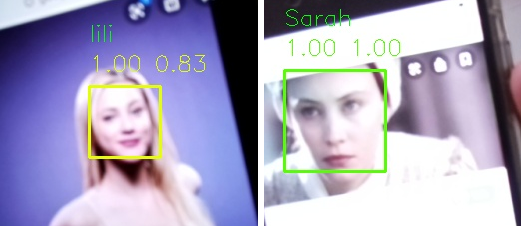


Fig 14 Detecciones obtenidas

**Conclusiones:**

Los resultados obtenidos fueron mayormente positivos. A diferencia de otros algoritmos de renacimiento de objetos, el reconocimiento facial no sobrecalentó el CPU de la Raspberry, la precisión fue siempre alta y el framerate (1.2FPS) es aceptable para la mayor parte de las aplicaciones del día a día.   
  
El programa en sí tiene varias funcionalidades que lo hacen más fácil de implementar, y se tuvieron en cuenta muchos factores tanto estéticos (colores de las cajas, información en la imagen), como de utilidad (un menú, la posibilidad de agregar usuarios con facilidad y también eliminarlos, archivar todos los registros tanto en texto como en imagen)

El proyecto fue hecho en un tiempo limitado, y podrían realizarse aún mejoras y modificaciones. El menú se podría realizar con una verdadera interfaz gráfica más interactiva, se podría exportar la base de datos a un formato más avanzado, y se podría jugar con cosas como la distancia euclidiana para mejorar la precisión.

En lo personal el proyecto me ayudó a mejorar mis conocimientos de Python, programación en general, conocimiento de Inteligencia Artificial y me dio un *insight* sobre el funcionamiento de los algoritmos utilizados hoy en día en el mercado.