

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería Informática

**SISTEMA DE RECONOCIMIENTO FACIAL PARA DISPOSITIVOS MOVILES Y ROBOT DE TELEPRESENCIA**

Alonso Rodrigo Serrano Alvarez

Dirigido por: Ángel Pérez de Madrid y Pablo

Codirigido por: Carolina Mañoso Hierro

Curso: 2017/2018



**SISTEMA DE RECONOCIMIENTO FACIAL PARA DISPOSITIVOS MOVILES Y ROBOT DE TELEPRESENCIA**

**Proyecto de Fin de Grado en Ingeniería Informática  
de modalidad *específica***

Realizado por: Alonso Rodrigo Serrano Alvarez

Dirigido por: Ángel Pérez de Madrid y Pablo

Codirigido por: Carolina Mañoso Hierro

Fecha de lectura y defensa:

**Resumen** 500 palabras max

**Palabras clave**

Visión artificial, reconocimiento facial, Android, Raspberry Pi, DIY, IoT, Python,

**Title**

**Abstract**

**Keywords**

Computer vision, facial recognition, Android, Raspberry Pi, DIY, IoT, Python,

**Índice**

[1. Introducción 5](#_Toc509518476)

[1.1. Alcance del proyecto 5](#_Toc509518477)

[1.2. Metas 5](#_Toc509518478)

[1.3. Consideraciones 5](#_Toc509518479)

[2. Hardware 6](#_Toc509518480)

[2.1. Elección del sistema 6](#_Toc509518481)

[2.2. Raspberry Pi 3 11](#_Toc509518482)

[2.3. Android Phone 13](#_Toc509518483)

[3. Reconocimiento facial 14](#_Toc509518484)

[3.1. Elección del sistema 14](#_Toc509518485)

[3.2. Servicio principal: Azure 19](#_Toc509518486)

[4. Software 28](#_Toc509518487)

[4.1. Introducción 28](#_Toc509518488)

[4.2. Elección del lenguaje 28](#_Toc509518489)

[5. Entrenamiento 29](#_Toc509518490)

[6. Conclusiones 30](#_Toc509518491)

[Bibliografía 31](#_Toc509518492)

**Lista de figuras y tablas**

1. Introducción
   1. Objetivo del proyecto

Este proyecto consiste en el estudio y desarrollo de un sistema de reconocimiento facial en plataformas portátiles de hardware para que pueda ser incorporado en otros proyectos o ser empleados de manera autónoma con otros fines.

La base de este proyecto es la filosofía del IoT (Internet de las cosas), en el que las partes del proyecto emplean Internet como infraestructura de operación.

Este proyecto se divide en 3 fases:

* Análisis: inicialmente se realiza una evaluación de las tecnologías existentes tanto de software como de hardware al tiempo que se declaran las características que se buscan en ellas. El fin es evaluar cuál de ellas pueden ser incorporada al proyecto. Los resultados de esta fase se encuentran documentados en los capítulos 2 y 3.
* Desarrollo: comenzará con el planteamiento de la estructura del proyecto y el comienzo de desarrollo de las partes individualmente. El desarrollo se documenta en el capítulo 4.
* Evaluación: una vez que las partes estén implementadas, se realizarán pruebas de cada componente individual y posteriormente pruebas del sistema en conjunto. Durante esta fase se realizarán los ajustes necesarios para alcanzar el objetivo del proyecto. Los resultados y conclusiones se estudian con mayor detalle en los capítulos 5 y 6.
  1. Metas

El sistema desarrollado debe buscar cumplir las siguientes características:

* Rapidez: se deben evitar largas esperas en el reconocimiento facial, debe ser tan rápido como posible
* Eficacia: se gestionará la información imprescindible, se deben evitar el envío, recepción o procesamiento de datos irrelevantes
* Fácil uso: los interfaces deben ser simples e intuitivos, se deben evitar situaciones complicadas para el usuario.
* Fácil incorporación: adaptar físicamente el sistema a una plataforma de hardware no debe suponer conexiones complejas ni conocimientos de electrónica, siguiendo la filosofía del Plug and Play (conectar y listo)
* Fácil mantenimiento: todo código que componga el programa debe ser de sencilla interpretación, bien documentado y sin el uso de clases o bibliotecas innecesarias.
* Bajo coste: deben buscarse componentes de bajo coste, pero sin olvidar la calidad y fiabilidad de los mismos.
  1. Estructura del proyecto

Una visión general del proyecto consiste en un sistema compuesto de 3 partes interconectadas mediante Internet como puede observarse en la Figura 1.

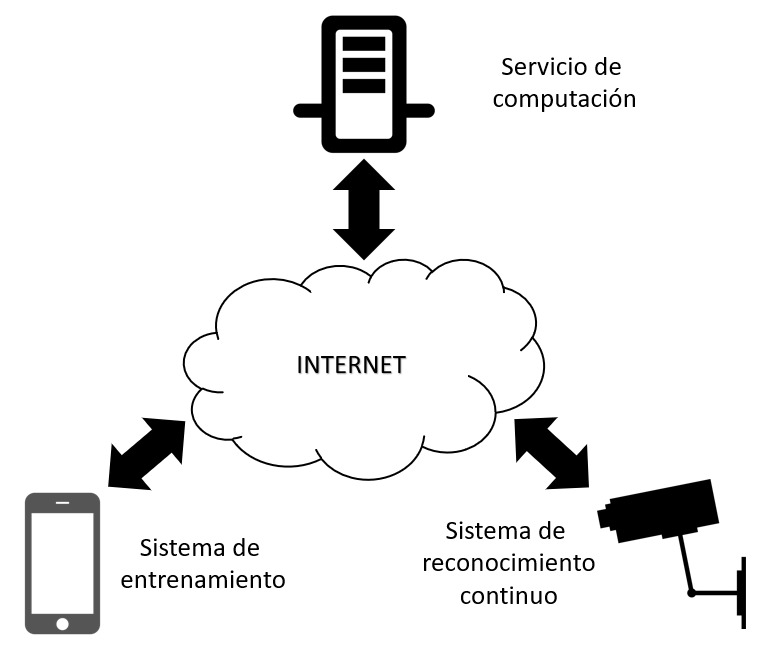


Figura 1 Esquema general del proyecto

Servicio de computación

El reconocimiento facial implica un alto número de operaciones y cálculos, y cuanto mayor es el número de individuos que el sistema es capaz de reconocer, mayor será el número de cálculos. Por tanto, los recursos que necesite un sistema de reconocimiento facial pueden ser considerables. El servicio de computación propio o ajeno se dedicará al almacenamiento de los datos de los individuos y el reconocimiento de caras aportadas por las otras partes del sistema. Toda la computación realizada debe ser transparente al usuario final.

Sistema de entrenamiento

Este componente permitirá a los usuarios finales entrenar el sistema mediante el envío simple de fotografías y nombres y recepción de mensajes simples. Será de uso simple y no debe requerir conocimientos de programación, la interfaz debe ser intuitiva y lo más simple posible.

Sistema de reconocimiento en continuo

El fin de este componente es la toma de imágenes a intervalos fijos, detección de caras e identificación de estas. El producto final será una imagen con las caras identificadas si las hubiera.

1. Hardware
   1. Elección del sistema

Una vez planteada la meta del proyecto, se comenzó la búsqueda de la plataforma de hardware adecuada para el uso previsto.

Las características deseables para el proyecto son las siguientes:

* Bajo coste
* Recursos (RAM, HDD) suficientes para cumplir las demandas del software
* Tamaño adecuado para cada parte del sistema
* Documentación amplia
* Soporte de la comunidad
* Integración de componentes

Las plataformas analizadas se describen a continuación:

Equipo de escritorio

La opción más versátil, los equipos de escritorio pueden presentarse de muchas formas, tamaños y características.

Altamente modulares, estos equipos permiten configuración a la carta para las características del sistema. Como se menciona en la introducción, una de las ideas iniciales fue un servidor capaz de hacer el servicio de reconocimiento facial, un equipo de escritorio era el candidato ideal para este servicio.

Para este proyecto este tipo de equipos tienen los recursos adecuados pero un tamaño excesivo que lo hace inviable para estar acoplado en un sistema de telepresencia u otro tipo de robot. El coste de este tipo de equipos tampoco son un factor favorable para su elección.

**Portátiles**

Comparado con los equipos de sobremesa, el portátil es más fácilmente acoplable a un sistema de telepresencia. Por el contrario, se sacrifica modularidad, los portátiles tienen menos capacidad de cambiar componentes. Si bien un portátil de gama media tiene recursos suficientes para el sistema, su precio y su dependencia de consumo de energía mediante el uso de baterías y/o cable de conexión a red eléctrica no lo convierten tampoco en un candidato viable.

**Tablets**

En algunos sistemas de telepresencia se han empleado tablets como el iPad con el fin de realizar un streaming constante. Los tablets tipo iPad son de poco peso y volumen aceptable, pero presentan ciertas limitaciones como el espacio de almacenamiento o la versatilidad inferior para crear y usar aplicaciones frente a otro tipo de tablets.



Figura 2 Sistema de telepresencia con iPad (Fuente robotshop.com)

Existen otros tipos de tablet como los basados en Android con mayor versatilidad para crear aplicaciones o los Surface con vastos recursos de hardware para implementar el sistema pudiendo usar Windows y Linux. Sin embargo, este tipo de tablets tienen un precio poco asequible y el consumo energético suele ser parecido al de un portátil.

**Teléfonos móviles**

Su pequeño tamaño les hace más viables aun que los sistemas anteriores, tienen gran cantidad de componentes integrados, cámara, Wifi, Bluetooth, 4G, etc, que lo hacen una opción viable. Sin embargo, según el tipo de móvil la programación es limitada a un tipo de lenguaje y el hardware puede ser insuficiente salvo que se adquiera un móvil de alta categoría.

Se descarto al teléfono móvil como núcleo del proyecto, pero no como una parte auxiliar al mismo, tras el análisis de las distintas plataformas se explicará con mayor detalle.

Existen plataformas basadas en 3 sistemas iOS, Android y Windows Phone.

No se puede valorar iOS para este proyecto pues que no se disponía de ningún dispositivo disponible para realizar pruebas.

Windows Phone es un sistema con poca presencia en el mercado y en breve estará obsoleto, puesto que Microsoft abandona la plataforma en favor de sistema más extendidos.

Android es el sistema para móviles más extendido, dispone de herramientas que facilita la programación, numerosa documentación y ejemplos y amplio soporte de la comunidad de programadores.

**Raspberry Pi 3**

[**https://www.raspberrypi.org/**](https://www.raspberrypi.org/)



Fig. 2 Raspberry 3 Modelo B (Fuente raspberry.org)

La familia de plataformas Raspberry son un conjunto de diferentes modelos de ordenadores de pequeño tamaño con gran cantidad de componentes integrados, Bluetooth, Wifi, HDMI, USB, etc.

A todos los efectos es un ordenador y puede usarse como tal, para navegar por Internet, emplear programas de ofimática, ver recursos multimedia y muchas más aplicaciones.

La principal ventaja de este sistema es su pequeño tamaño (cabe en la palma de la mano) que permite acoplarlo fácilmente a casi cualquier tipo de robot y su bajo consumo energético permite acoplarlo a la batería del robot sin que suponga un consumo excesivo.

Todo el software que emplea es open source, lo que permite la instalación de diferentes sistemas operativos y programas. Incluye el uso de una versión de Linux llamada Raspbian muy versátil y de fácil uso y dispone de ciertos intérpretes de lenguaje y compiladores ya integrados para su uso (Python, C)

El coste de esta plataforma es muy bajo, en torno a 40€ lo cual lo hace muy asequible para este proyecto.

Existe una gran cantidad de documentación y soporte de la comunidad de desarrolladores, además de gran cantidad de ejemplos sobre proyectos que pueden realizarse con esta plataforma, desde servidores web hasta cajas fuertes bloqueadas por GPS.

Otra ventaja es que dispone de varios puertos USB para poder conectar distintos componentes, una conexión para una cámara exclusiva para la plataforma y pines GPIO que le permiten conectar con otras plataformas o componentes como chips de radio RF, pantallas externas, sensores, efectores, etc.

El principal inconveniente es su limitación de recursos, el modelo B tiene 1 Gb de RAM compartido con la GPU y un procesador de 1.2 Ghz.

**Arduino**

[**https://www.arduino.cc/**](https://www.arduino.cc/)



Fig. 3 Arduino UNO (Fuente android.cc)

Es una familia de plataformas de hardware open source que permiten crear proyectos de hardware y software fácilmente empleando el lenguaje C y conexión sencilla de sensores y efectores. El precio y consumo de los Arduino es muy bajo.

Sin embargo, es muy limitado en cuanto a recursos de memoria, haciendo imposible que sirva para el sistema, si bien es cierto que se puede emplear como parte un robot de telepresencia esto queda fuera del alcance de este proyecto.

**Otras plataformas**

Existen en el mercado otra serie de plataforma parecidas a Raspberry Pi:

* ASUS Tinker Board
* Banana Pi M3
* C.H.I.P. Pro Dev Kit
* Intel Joule
* NanoPC-T3
* NanoPi Neo Plus 2
* ODroid Xu4
* Orange Pi Plus 2E
* Pine A64

Las variaciones son respecto a precio, memoria, velocidad de procesador, componentes integrados. Algunos como el Intel Joule ofrecen características muy interesantes para este proyecto, pero con precios muy elevados.

No obstante, ninguno de los citados anteriormente tiene la documentación y soporte como Raspberry Pi.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Equipo escritorio** | **Portátil** | **Tablet** | **Teléfono móvil** | **Raspberry 3 B** | **Arduino** |
| **Precio** | ALTO | ALTO | ALTO | MEDIO | BAJO | BAJO |
| **Tamaño** | GRANDE | MEDIANO | MEDIANO | PEQUEÑO | PEQUEÑO | PEQUEÑO |
| **Documentación** | VARIADA | VARIADA | ESCASA | VARIADA | ABUNDANTE | ABUNDANTE |
| **Recursos** | ALTO | ALTO | ALTO | MEDIO | MEDIO | BAJO |
| **Soporte** | ALTO | MEDIO | BAJO | MEDIO | ALTO | ALTO |
| **Consumo** | ALTO | ALTO | MEDIO | MEDIO | BAJO | BAJO |
| **Modularidad** | ALTO | BAJA | BAJA | BAJA | MEDIA | MEDIA |
| **Integración** | MEDIA | ALTA | ALTA | ALTA | ALTA | MEDIA |
| **Conectividad** | ALTA | ALTA | BAJA | BAJA | ALTA | ALTA |

Tabla 1 Comparación entre plataformas de hardware

* 1. Raspberry Pi 3

Esta plataforma se elige finalmente por estar a mitad de camino entre el tamaño y consumo de un móvil y los recursos y versatilidad de un ordenador.

La gran cantidad de documentación sobre la Raspberry y su precio son una ventaja adicional apreciada.

Se analizarán diversos aspectos de la Raspberry antes de comenzar con su uso.

Configuración del sistema

La configuración específica y paquetes instalados en la raspberry para este proyecto se detallan en el anexo #

Cámara integrada vs. webcam

Lógicamente para la captura de imágenes se necesitará una cámara digital, en la Raspberry se pueden conectar una webcam mediante USB o bien emplear el conector de cámara integrado en la placa base.

El uso de una webcam tiene como ventajas:

* existencia de webcams de alta definición y gran número de megapixels
* incorporación de flash de algunos modelos
* trípode u otro tipo de soporte incorporado

sus desventajas son:

* necesidad de drivers específicos que no siempre existen para Linux
* necesidad de módulos específicos para incorporar su uso en lenguajes de programacion

La cámara de la propia Raspberry tiene como ventajas:

* conector específico dedicado en la propia placa
* tamaño reducido
* Raspbian incorpora drivers de serie para su uso fácil
* existen módulos para su fácil uso en programación

Su desventaja principal es que al ser poco mas que una placa de circuito no incorpora soporte de serie y en ocasiones es difícil de colocar en una posición correcta y estable, lo cual le da más fragilidad.

Por tanto, se decide adquirir una cámara propia de Raspberry modelo Module 2 cuyas características más relevantes son sus 8 Megapixels y video en 1080p. [ref]



Fig. 4 Cámara de Raspberry Pi

Una vez conectada el método para configurarla es muy sencillo:

En una ventana de terminal se ejecuta el comando

sudo raspi-config

Se selecciona “Enable camera”, posteriormente “Finish” y se procede a realizar un reinicio del sistema. Una vez concluye el reinicio, se puede comprobar si el sistema funciona empleando este simple comando en el terminal

raspistill -v -o test.jpg

que tomará la foto y la guardará en el archivo test.jpg

Otros componentes

Para poder usar la Raspberry correctamente se emplean otros componentes de los cuales no es necesario entrar en detalle ni requieren una configuración complicada. Estos componentes son: pantalla, teclado, ratón, conexión Wifi y conexión Bluetooth.

* 1. Android Phone

Como se ha explicado anteriormente un teléfono móvil no es suficiente para ser la parte acoplada al sistema de telepresencia, sin embargo, puede ser altamente útil para entrenar el sistema o incluso para poder tener el sistema de reconocimiento de manera portable.

La elección de Android es debida a la gran extensión de este sistema en multitud de diversos modelos de teléfonos y tablets, por tanto, una vez generada la aplicación se puede instalar en diversos dispositivos que podrán realizar la misma función.

Otro de los motivos es que el IDE Android Studio permite una forma fácil de creación de aplicaciones Android y existe gran cantidad de documentación al respecto.

Como base para la aplicación se empleará un Samsung Galaxy S3 para poder desarrollar la aplicación y realizar las pruebas.

1. Reconocimiento facial
   1. Elección del sistema

Durante la fase de análisis del proyecto, se encontraron números servicios de reconocimiento facial que podían cubrir las necesidades de este proyecto, cada servicio disponía de características diferentes lo cual hizo necesario un análisis profundo antes de determinar cuál era el correcto.

Las características deseables para elegir el servicio son:

* Facilidad de uso
* Documentación madura sobre el servicio
* Compatibilidad con los lenguajes y tecnologías de este proyecto
* Soporte de la comunidad
* Plan de estudiante o plan gratuito durante un periodo de tiempo

Se describen brevemente los servicios candidatos y en mayor detalle los finalistas.

**#####CANDIDATOS**

**Amazon Rekognition**

<https://aws.amazon.com/rekognition/>

Forma parte del ecosistema AWS, lo cual permite fácil integración con otras aplicaciones que se desarrollen empleando dicho ecosistema. Dispone de un plan gratuito durante 12 meses con limitaciones. El plan incluye reconocimiento facial con limitación de numero de caras a reconocer al día.

El problema de este servicio es que la documentación es bastante limitada para Python y casi no existen ejemplos.

**Face++**

<https://www.faceplusplus.com/>

Además del servicio basado en la nube dispone de un servicio offline para Android, en principio esta opción era interesante, pero la opción offline solo incluye detección de caras y no el reconocimiento de estas.

Incluye un plan gratuito, aunque las limitaciones del plan solo permiten realizar una transacción por segundo, en ocasiones esto puede ralentizar el sistema.

No obstante, al intentar crear un usuario para probar la cuenta gratuita y poder ver las condiciones de uso, la web se ha quedado atascada en todo intento.

**Kairos**

<https://www.kairos.com>

Este servicio ofrece valor añadido como detección de edad, genero, estado emocional y análisis de video en tiempo real.

El plan gratuito ofrece acceso a todas las características con limitaciones, una de ellas es 25 transacciones por minuto y 1500 en total al día. Parece insuficiente para un reconocimiento continuo de caras y menos adecuado para análisis en tiempo real de video.

**Google Vision**

<https://cloud.google.com/vision/>

El servicio integrado con Google Cloud ofrece muchas posibilidades, pero el reconocimiento facial no es una de ellas. Por tanto, se descarta para este proyecto como servicio principal

Durante la fase de implementación se descubrió que emplear OpenCV para la detección de caras en el teléfono móvil conlleva un coste de recursos y de implementación alto.

Si bien el servicio de nube no se empleará en el sistema, un subconjunto llamado Mobile Vision dispone de una biblioteca con métodos que permiten una detección de caras más rápida.

**#####FINALISTAS**

**IBM Watson**

<https://www.ibm.com/watson/>

En principio se planteó emplear este servicio para el proyecto debido a la robustez y fama de Watson.

El plan gratuito incluye solo la clasificación de 250 imágenes al día, sin embargo, el plan de estudiante permite un acceso mayor durante 6 meses.

La documentación podría ser mejorable y necesita aportar más ejemplos, pero con el material publicado se puede empezar a crear un proyecto viable.

Aunque explícitamente Watson no tiene un sistema de reconocimiento facial, este sistema quedo finalista puesto que existía una manera de crear dicho sistema.

Watson permite la creación de clasificadores específicos en los que se pueden almacenar fotos, entrenar el sistema y posteriormente comparar una foto con cada clasificador y obtener un porcentaje de confianza.

Las pruebas iniciales y facilidad de uso de la API cumplían con las expectativas para este proyecto, de hecho, en el entrenamiento “Bill Gates vs. Ben Linus” que se verá en el capitulo de este documento dedicado al entrenamiento, los porcentajes de confianza obtenidos eran muy altos (hasta 82%).

El hecho de no tener un sistema especifico de reconocimiento facial parecía no ser un problema al poder crear clasificadores individuales y cada uno ligarlos a una persona. Además, cada clasificador puede alimentarse también con ejemplos negativos, se planteo que por cada la primera foto de una persona que se subiese como ejemplo positivo al clasificador se añadiese como ejemplo negativo a los demás clasificadores.

Tres motivos hicieron que finalmente se descartase el sistema como núcleo del proyecto:

* El requisito mínimo de fotos para un clasificador es de 10, no siempre es posible disponer ese número de imágenes, por ejemplo, en el momento de entrenar al sistema mediante el móvil es posible que la persona a la que se fotografía no tenga la paciencia ni ganas de esperar a que se le realicen 10 fotos. Emplear la misma foto 10 veces no es una buena factible, no permite un buen entrenamiento del clasificador y baja el porcentaje de confianza obtenido.
* Durante el periodo de prueba de Watson el sistema migro el tipo de cuenta que se empleaba a cuenta “Lite” con la funcionalidad muy limitada, los motivos de esta migración se desconocen puesto que la cuenta tenia 6 meses de uso de estudiante. Si bien es cierto que durante esta transición aparentemente IBM estaba cambiando partes de su sistema y pudo estar relacionado con la migración de cuentas
* El tercer motivo fue que el clasificador no acepta nuevas fotografías una vez creado y subido las imágenes iniciales de entrenamiento, esto crea un problema, puesto que cada vez que se quiera añadir una fotografía al clasificador especifico de una persona se tendría que eliminar completamente obtener las imágenes iniciales, añadir las nuevas, recuperar los ejemplos negativos, añadirlo también y subir toda la información junta de nuevo, esto supone una sobrecarga de transacciones y mayor lentitud global del sistema.

Si bien los dos primeros motivos podrían solucionarse, el tercero no tiene una solución viable. Por tanto, a mediados de febrero se comenzó a probar en mayor medida el sistema Azure y a principios de marzo se abandono completamente el sistema Watson. Esto motivo a un cambio en el planteamiento del proyecto y tener que comenzar a codificar de nuevo el sistema.

**Microsoft Azure**

<https://azure.microsoft.com>

Azure incluye un servicio de reconocimiento facial con posibilidad de creación de grupos (amigos, trabajo) y opciones de valor añadido como detección de emociones, gafas, genero, estimación de la edad, etc.

La documentación es bastante completa pero los ejemplos en Python al menos no son muy adecuados.

Microsoft ofrece un plan gratuito de 30 días para cuentas nuevas y hasta 12 meses para estudiantes, existen ciertas limitaciones como el empleo de 10 llamadas por segundo a la API, pero son más que suficientes para el desarrollo de este proyecto.

Las primeras pruebas devolvían un porcentaje de confianza algo inferior a la obtenida con Watson en las pruebas “Bill Gates vs. Ben Linus” (hasta un 73%).

El sistema se dejo de lado hasta que surgieron los problemas con Watson detallados anteriormente, a mediados de febrero se comenzaron pruebas con el sistema Azure y se ha seguido manteniendo para finalizar el sistema.

**OpenCV**

<https://opencv.org>

No es un servicio como los anteriores, sino una biblioteca abierta con algoritmos que se pueden emplear para proyectos de visión artificial y aprendizaje automático.

En principio era una opción muy deseable, dado que dispone de gran cantidad de algoritmos muy robusto, revisados y comprobados. Sin embargo, la limitación del hardware como se expuso en el capitulo dedicado a ello, hace que este sistema no sea factible a largo plazo. Cuantas más caras formen parte del sistema, más se tardará en el entrenamiento. El hardware elegido es muy limitado y no funcionará bien pasado cierto numero de personas y/o caras.

Es gratuita, no siendo necesario ningún tipo de suscripción, pero no ofrece un servicio explícito de reconocimiento facial, hay que construirlo.

En principio queda descartado como núcleo de este proyecto, pero tiene un valor añadido que le hace quedar como finalista como servicio de apoyo.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| **Servicio online** | SI | SI | SI | SI | SI | SI | NO |
| **Servicio offline** | NO | SI | NO | NO | NO | NO | SI |
| **Reconocimiento facial exclusivo** | SI | SI | SI | NO | NO | SI | NO |
| **SDK** |  |  |  | SI | SI | SI |  |
| **API** |  |  |  |  | SI | SI |  |
| **Plan estudiante** | NO | NO | NO | NO | SI | SI | SI |
| **Plan gratuito** | Limitado | Limitado | Limitado | Limitado | Limitado | Limitado | SI |
| **Servicios adicionales (almacén, bases de datos)** | SI | SI | SI | SI | SI | SI | NO |
| **Documentación** |  |  |  |  |  |  |  |
| **Valor añadido (detección de emociones, rasgos, sexo)** |  |  |  | NO | NO | SI |  |

Tabla 2 Comparativa entre servicios de reconocimiento facial

* 1. Servicio principal: Azure

Tras darse de alta en el servicio, como estudiante en el caso de este proyecto, el panel principal de Azure permite elegir un servicio para comenzar a trabajar. Para este proyecto se elige el servicio QeQ Cognitive Services que permitirá trabajar con la API Face. Una vez dado de alta el servicio Azure entrega una clave de suscripción de 32 caracteres con la que se deben realizar todas las consultas y llamadas a la API.

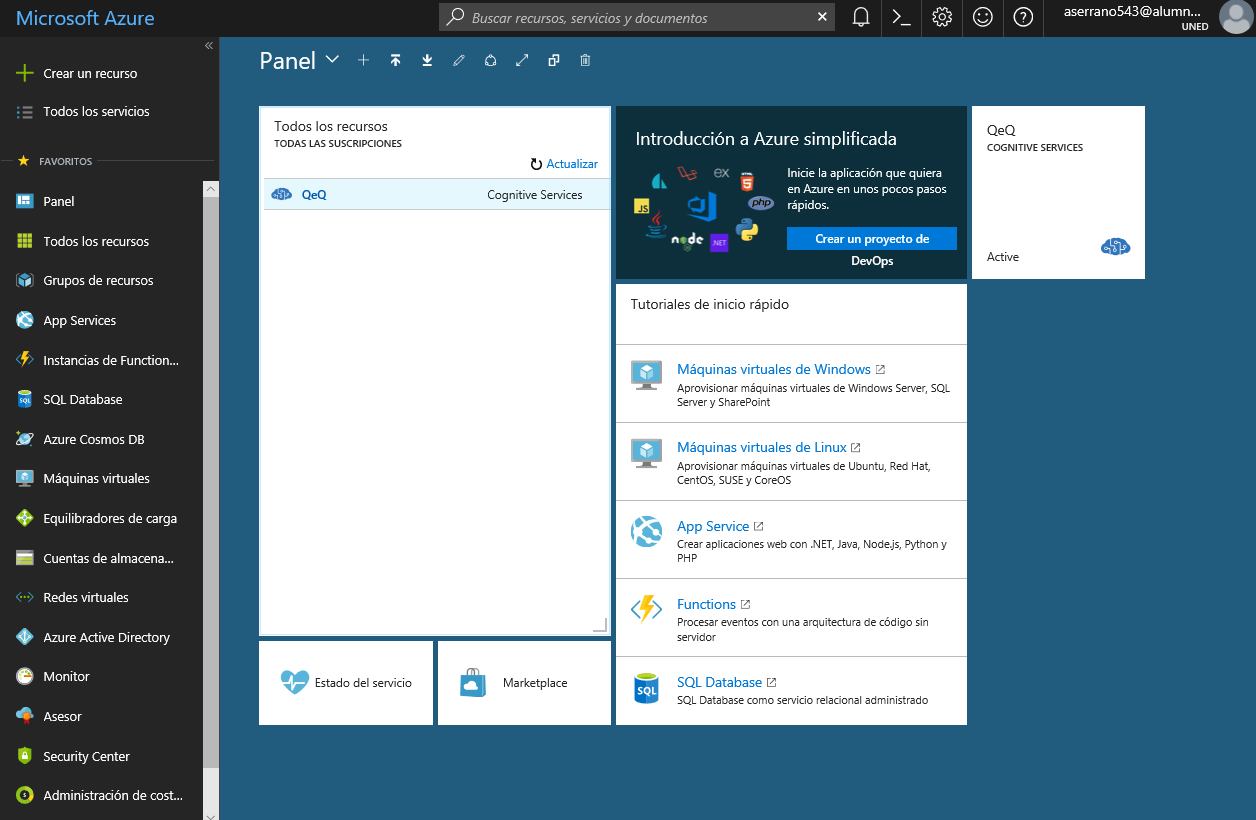


Fig. 5 Panel de Azure tras alta del servicio QeQ

Para el reconocimiento facial Azure dispone del servicio Cognitives Services. Este servicio dispone de un API bien documentado llamado Face API y de varios SDK para Android, iOS, Python y Windows de libre acceso disponibles en Github.

La base de la API es la construcción de una dirección URL que combina varias partes:

Un método HTTP: POST, GET o DELETE. Se usa uno u otro dependiendo de la función del API que se quiera emplear

Una dirección base que indica el servidor y servicio que se quiere acceder:

<https://northeurope.api.cognitive.microsoft.com/face/v1.0/>

Una parte adicional que especiﬁca que parte del servicio se solicita, grupo de personas, persona, etc.

/persongroups/{personGroupId}/persons/{personId}/persistedFaces[?userData][&targetFace]

Una cabecera con información sobre el contenido del cuerpo del mensaje (generalmente formato JSON) y la clave de subscripción.

headers = {'Content-Type': 'application/json', 'Ocp-Apim-Subscription-Key': '{subscription key}', }

Un cuerpo del mensaje con distinta información que irá con la petición HTTP

* parámetros (ej, umbral de confianza)
* información (en formato JSON)
* datos (imagen)

La API se compone de varias partes y tiene muchas aplicaciones como la identificación de grandes grupos de persona. Dado el objetivo de este proyecto, solo algunas partes de la API serán empleadas. Siendo estas Face, PersonGroup y Person de las que solo se describirán los métodos generales que se emplearán.

En líneas generales se pueden crear varios grupos de personas (PersonGroup) cada uno es capaz de almacenar personas (Person), cada Person a su vez puede almacenar caras (Face) y de esta manera los datos quedan fácilmente almacenados.

Cada entidad tiene su identificador único que permite recuperar rápidamente la información pertinente. Los 3 que se emplearan son:

* faceId: identifica una cara
* personId: identifica una persona
* personGroupId: identifica un grupo

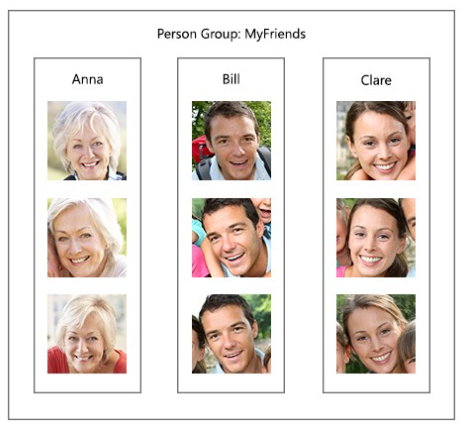


Fig. 6 Ejemplo de organización de entidades. (Fuente Microsoft)

**Face**

Dispone de un método **Detect**, que detecta caras en una imagen, pero por eficiencia no se empleará, esa tarea se derivará al OpenCV y se describirá más adelante.

El método que se empleará es **Identify**, este método identificará caras, para ello como parámetro de entrada se introduce:

* un faceId (pueden ser varios) con la cara que queremos identificar (esta cara tiene que estar en el sistema). En la sección de implementación se explicará mejor como realizar esta operación
* un groupPersonId para identificar el grupo en el que se comparara la cara. Azure permite comparar una cara con un grupo entrenado por llamada, aunque sería fácil implementar un código para verificar todos los grupos a la vez en varias llamadas
* un número máximo de candidatos de retorno (este parámetro es opcional, por defecto su valor es 10)
* un umbral de confianza, el parámetro es opcional, sirve para que no devuelva resultados cuyo porcentaje de confianza este por debajo del valor indicado

El retorno de **Identify** serán datos en formato JSON indicando los personID de las personas a las cuales podría pertenecer la cara objeto de identificación y la confianza de estas.

**Person**

Dispone de métodos para agregar, recuperar y borrar. Se emplearán principalmente 3 de ellos.

**Add Face** se encarga de añadir las caras (siempre de una en una) a la persona que se le indique, para ello como parámetro de entrada se introduce el personId de la persona objetivo, el groupPersonId donde este situada la persona y una URL con la imagen que contiene la cara. Opcionalmente se puede añadir las coordenadas del rectángulo que delimita donde está la cara situada dentro de la imagen. Si la llamada es exitosa se obtiene el faceId correspondiente.

**Create** es un método sencillo que aportando un groupPersonId y un nombre para la persona que se desea crear, crea una persona y devuelve su personId

**Get** se emplea para conseguir los datos de una persona a partir de su personId

**PersonGroup**

De forma análoga a Person dispone de métodos para agregarm recuperar y borrar. Dispone de un método créate que funciona de forma parecida a Person, tan solo aportando el nombre del grupo, este es creado y devuelve el groupPersonId correspondiente.

Existen dos métodos que se emplearan y son importantes.

**Train**: tras subir fotos a una persona dentro de un grupo, es necesario entrenar al clasificador, por una parte, se entrena el clasificador individual y por otra parte se entrena como grupo. Esto sirve por si desean crean grupos con características similares para otro tipo de aplicaciones, como clasificar gente joven y gente mayor o gente con el pelo corto y pelo largo, etc.

Para este proyecto se necesitan entrenar los clasificadores individuales por tanto cada vez que se añada una cara a una persona será necesario invocar el entrenamiento.

Como parámetro de entrada se necesita el groupPersonId del grupo objetivo y si el entrenamiento ha sido correcto se recibe un mensaje JSON vacío.

**Get training status:** el entrenamiento es más complejo cuantos mas datos se tengan, por tanto, a medida que crece el sistema el tiempo de espera para la finalización será mayor. Por tanto, es prudencial usar este método para comprobar si se ha termina el entrenamiento y es seguro continuar usando el sistema.

El parámetro de entrada será el groupPersonId del grupo objetivo de verificación y se obtendrán 4 cadenas de texto con información sobre el entrenamiento:

* estado del entrenamiento (no comenzado, ejecutando, terminado o fallido)
* fecha y hora de creación del grupo
* fecha y hora de última acción realizada sobre el grupo
* mensajes de error si los hubiera

Con los objetos y métodos vistos anteriormente es suficiente en principio para poder crear una implementación propia se podrán ver en el capítulo dedicado a la implementación.

Consideraciones

La clave será omitida del código fuente y de las imágenes, puesto que es una clave privada que se emplea en otros proyectos.

El autor de este proyecto actualmente reside en Suecia, por este motivo el servicio empleado de Azure puede que requiera una configuración distinta en España. El servicio de Azure emplea el servidor

<https://northeurope.api.cognitive.microsoft.com>

mientras que en España debería emplearse el servidor

<https://westeurope.api.cognitive.microsoft.com>

En el código fuente se contempla la posibilidad de cambiar este parámetro para que sea fácil configurarlo para otras regiones.

**3.3 Servicio auxiliar: OpenCV**

Como se vio en la parte de análisis y en el capitulo dedicado a hardware, OpenCV requiere mayor uso de recursos cuanto mas complejo sea el sistema y debido al hardware elegido, los recursos son muy limitados.

Este sistema dispone de varias funciones de interés, con el fin de evitar la sobrecarga y el envió eficaz de imágenes se empleará OpenCV en una parte del sistema.

Este proyecto pretende ser móvil, para ello las comunicaciones a Internet serán inalámbricas (wifi o uso de datos de telefonía móvil) por tanto es importante que los envíos sean ligeros.

El empleo de OpenCV puede ayudar a resolver una serie de inconvenientes causados al tomar fotos, ya que estas al ser de alta resolución pueden ocupar unos cuantos megabytes y las consecuencias del tamaño:

* El envió de datos mediante medios inalámbricos generalmente tienen menor velocidad que si se empleasen conexiones físicas, puede suponer una ralentización del sistema
* En el caso del uso de redes de telefonía móvil se añade el coste por megabyte enviado, el sistema no será tan económico como se pretende
* Al recibir imágenes grandes el servicio de reconocimiento tiene que detectar primero las caras, separarlas y procesarlas, esto crea más tiempo de procesamiento en el servicio, disminuyendo el rendimiento global del proyecto.

Por tanto, la solución es el envió de imágenes que:

* contenga la información justa y necesaria, es decir, solo la cara
* solo contengan una cara, disminuyendo el tiempo de procesamiento
* sean de menor tamaño

Para ello se emplearán un conjunto de algoritmos de OpenCV que, tras la toma de la foto, recortaran las caras relevantes para su envió individual al servicio y posterior identificación de cada una de ellas.

Si bien se ha comentado la limitación del hardware, la implementación del recorte de caras es ligera, rápida y no demanda demasiados recursos.

OpenCV dispone de un módulo llamado **objdetect**, (Object Detection) que contiene la clase CascadeClassifier.

CascadeClassifier está basado en la cascada de clasificadores, este método se emplea ampliamente para la detección de caras debido al bajo consumo de recursos.

Una breve introducción general teórica es necesaria para entender porque se elige este método.

Para la detección de caras se emplea en primer lugar una generación de candidatos basado en el uso de una ventana deslizante de diversos tamaños que se aplica a diferentes partes de la imagen, cada ventana aplica un filtro que determina si es buen candidato o no.

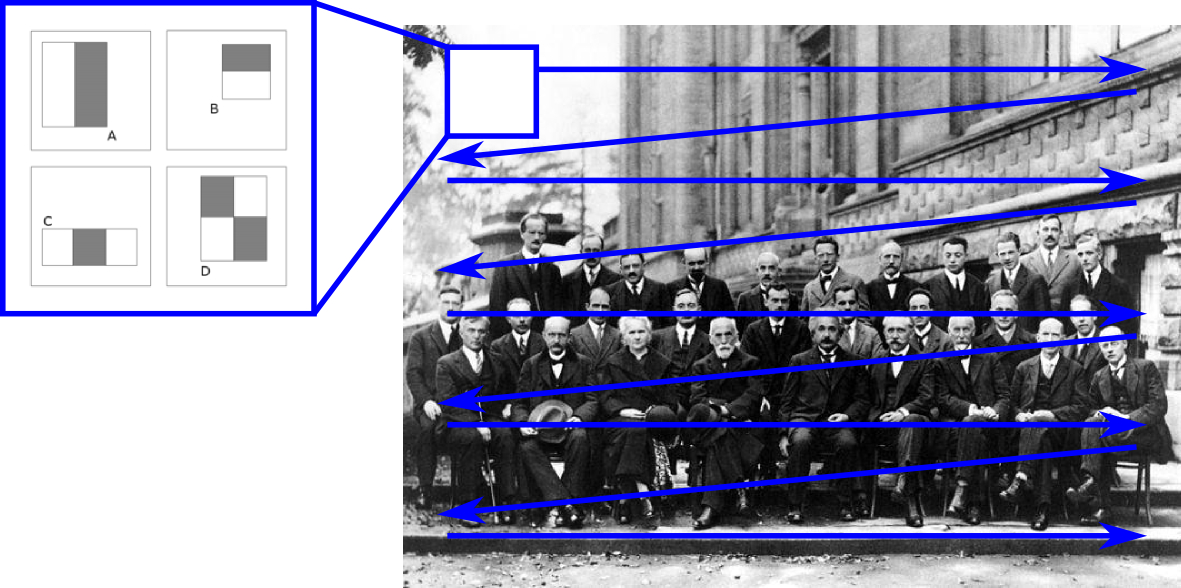


Fig. 7 Ejemplo de ventana deslizante (Fuente Wikipedia)

Tras la generación de candidatos se realiza la clasificación, cada ventana se somete al clasificador y este devuelve correcto o incorrecto.

El CascadeClassifier que va a emplearse funciona en líneas generales de esta manera:

Para el filtrado se emplean filtros Haar en diferentes escalas y en múltiples posiciones

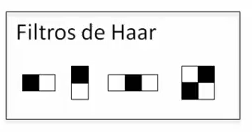


Fig. 8 Ejemplo de filtros de Haar (Fuente Coursera)

Tras la aplicación de los filtros Haar en las ventas deslizantes se generan los candidatos y se almacenan.

Posteriormente cada candidato se somete a una cascada de clasificadores, el objetivo del sistema es que el candidato pase cada clasificador individual y si uno de ellos no lo clasifica, entonces el candidato se descarta automáticamente. De esta forma se incrementa la eficacia, puesto que, al estar en serie, si un clasificador rechaza al candidato no se requieren mas comprobaciones.

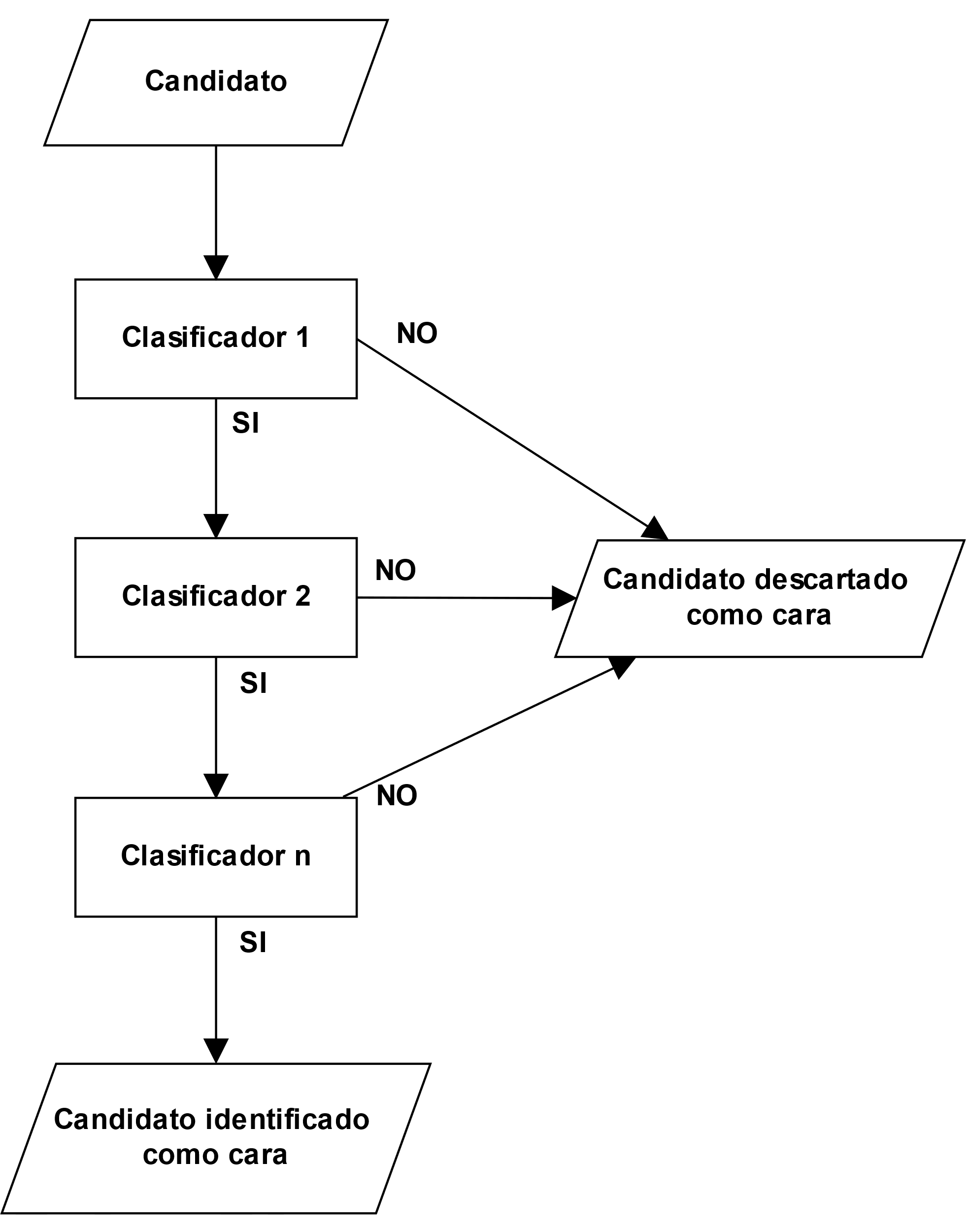


Fig. 9 Cascada de clasificadores (Fuente propia)

Junto a CascadeClassifier se pueden emplear también los LBP (Local Binary Patterns) como alternativa a los filtros Haar.

El motivo principal de usar filtros Haar frente a los LBP es la precisión, LBP es más rápido, pero menos preciso

Para este proyecto se quiere rapidez, sin embargo, la detección de caras para esta parte del sistema no genera ningún tipo de almacenamiento ni creación de clasificadores, ni entrenamiento, por tanto, la ganancia de velocidad es menos relevante que la precisión.

Se emplearán los filtros Haar en combinación con la clasificación en cascada.

1. Software
   1. Introducción

El sistema se divide en dos partes:

FaceRecon: Reconocimiento continuo de caras. Instalado en la Raspberry con un programa que continuamente reconoce caras y devuelve una imagen y texto con las identificaciones de las caras.

FacePal: Entrenador de bolsillo. Se instalará en un teléfono móvil y permitirá el entrenamiento de una manera más cómoda y opcionalmente el reconocimiento de caras.

* 1. Elección del lenguaje

Aunque existan una gran variedad de lenguajes y sería interesante explorar varios candidatos para elegir el que mejor se adapta al proyecto, las limitaciones establecidas por el hardware y servicios elegidos acotan bastante la lista de lenguajes posibles a emplear.

FaceRecon

Al estar instalado dentro de la Raspberry, existen muchas opciones de elección del lenguaje. La primera pregunta para poder realizar la criba es ¿Lenguaje compilado o interpretado?

El lenguaje compilado, como C++, tiene la ventaja de mayor rapidez, en Raspian pueden instalarse fácilmente varios compiladores de C++ (g++ o clang), pero a medida que crece el proyecto el enlazado con otras bibliotecas de software se puede volver complicado y confuso, además es responsabilidad del programador evitar las violaciones de acceso (comúnmente denominado segfault) y en ocasiones se tarda mucho en averiguar donde esta el problema en el código. Dado el margen de tiempo del que se dispone se descarta C++ como lenguaje elegido.

Existen otros lenguajes como Rust que son compilados y quitan al programador la responsabilidad de muchas comprobaciones. Sin embargo, en el momento de la redacción de este proyecto Rust es un lenguaje relativamente nuevo y aun no muy extendido como otras opciones.

Los lenguajes interpretados requieren el uso de un interprete cada vez que se ejecuta el código, esto hace que el rendimiento global sea inferior al de un código compilado. Sin embargo, permite la creación de aplicaciones más agiles puesto que al ejecutar el interprete se encarga de todo, no es necesario compilar y luego ejecutar.

Los dos lenguajes que serian adecuados para programar esta aplicación serian Java y Python. Si bien ambos son muy extendidos y bien documentados, se decide emplear Python (versión 3) por su facilidad de uso y facilidad de lectura del código, además Python permite que una función devuelva más de un valor incluso de distintos tipos o permite colecciones de objetos de distinto tipo sin tener que crear clases exclusivas para este fin.

**FacePal**

Al estar destinado a que sea instalado en un sistema Android, la opción más lógica es el lenguaje Java. Android dispone de un IDE gratuito llamado Android Studio que permite la programación de aplicaciones para Android empleando el lenguaje Java con muchas funciones integradas.

Para el diseño estético de la aplicación también se empleará XML, si bien no es un lenguaje de programación como tal, es necesario entenderlo para hacer un diseño correcto de la aplicación.

**Consideraciones previas**

Se ha optado por usar inglés como idioma para determinar el nombre de variables, funciones y otras piezas de software. Sin embargo, en esquemas y comentarios del código se empleará español.

* 1. FaceRecon

La descripción genérica de la aplicación sería la expuesta en la Figura 3.

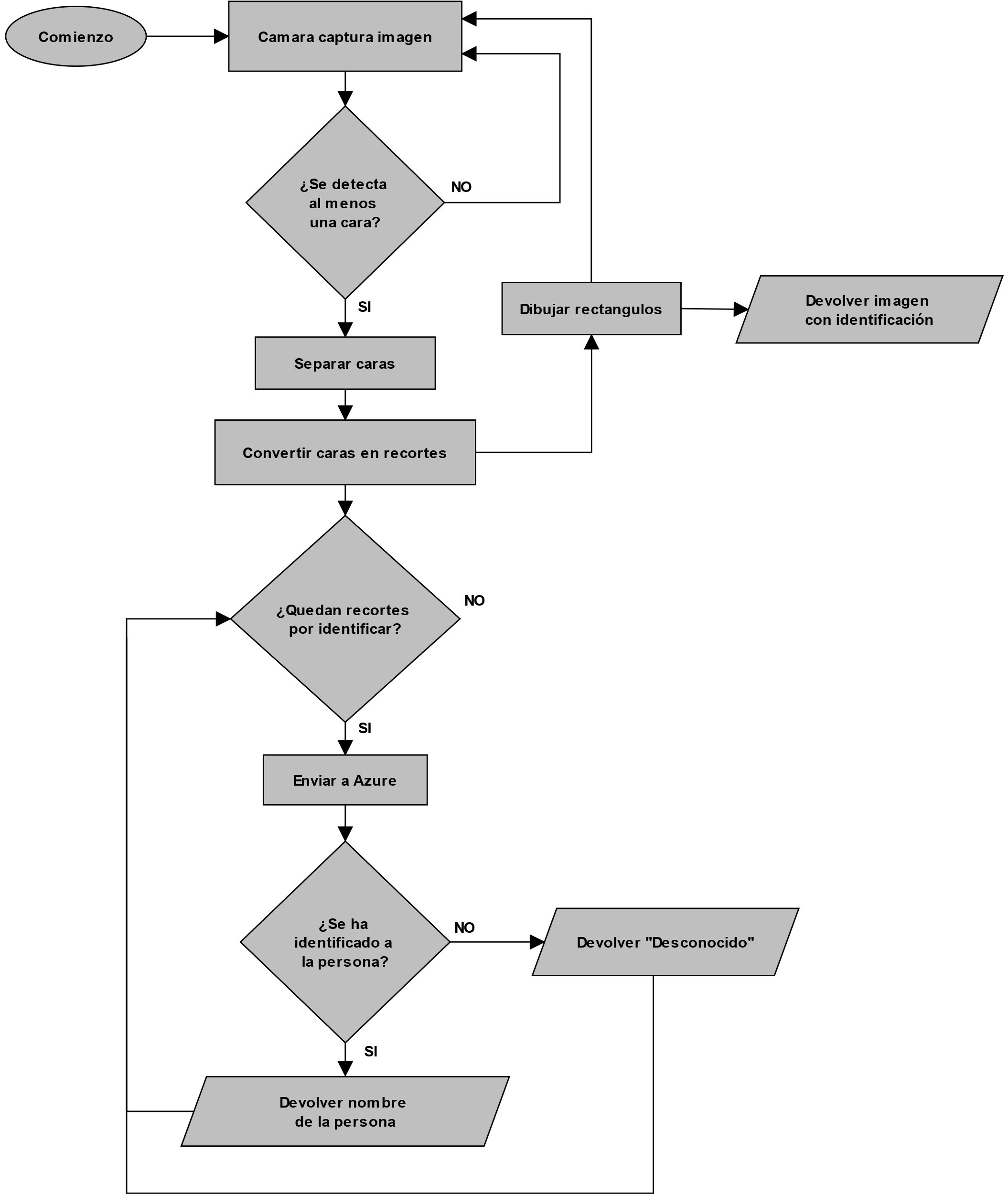


Figura 3 Descripción de FaceRecon

Para facilitar la comprensibilidad del sistema y el mantenimiento se dividirá el código en módulos/clases agrupando funciones similares, en la Figura 4 se puede observar la dependencia de clases/modulos.

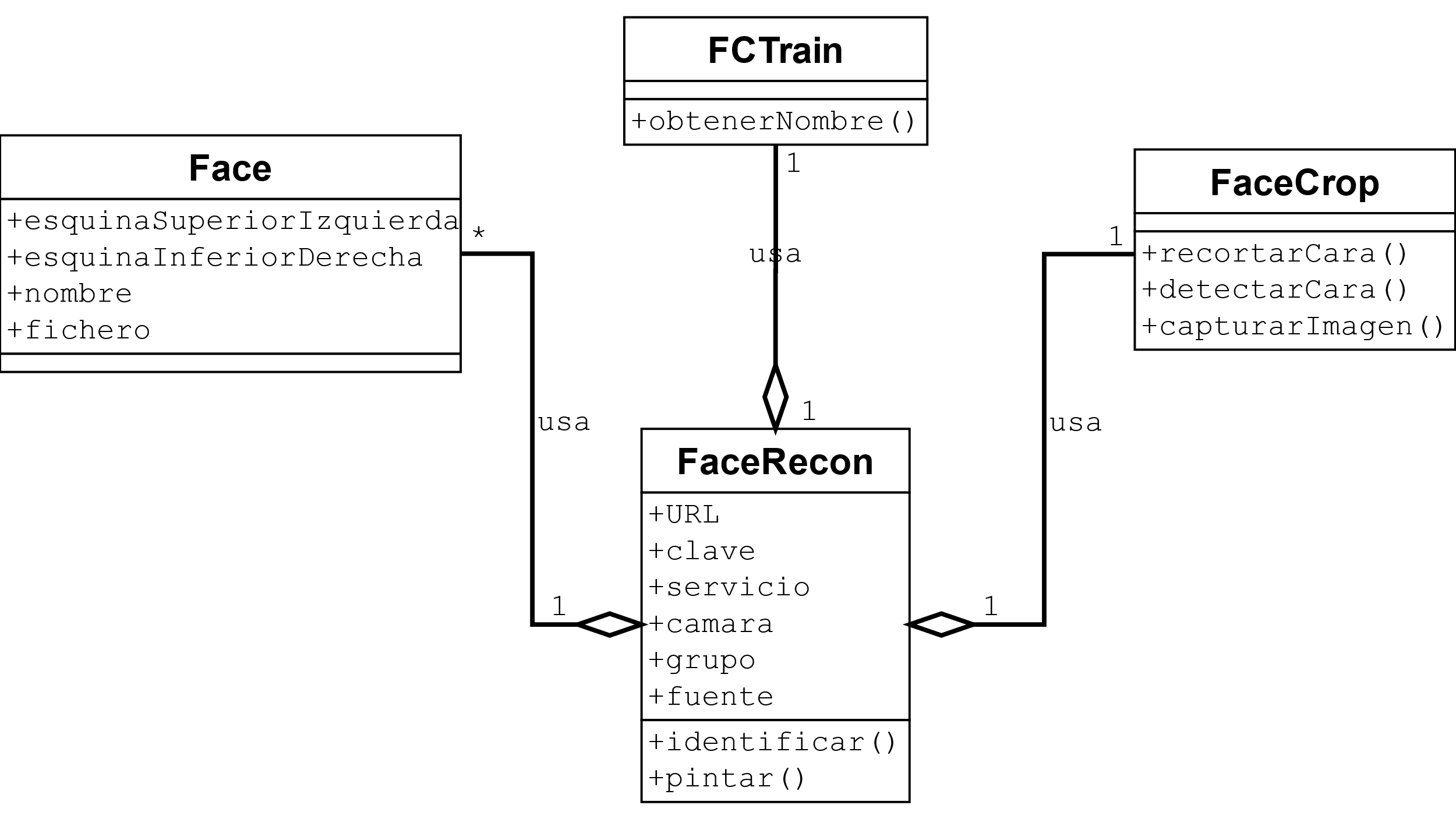


Figura 4 Diagrama de clases de faceRecon

Face

Clase que define la estructura de datos que contiene la información de cada cara.

Su constructor requiere que se le pasen dos parámetros ulCorner y lrCorner, que corresponden con las coordenadas de las esquinas del rectángulo que contiene la cara objetivo como se muestra en la Figura 5.

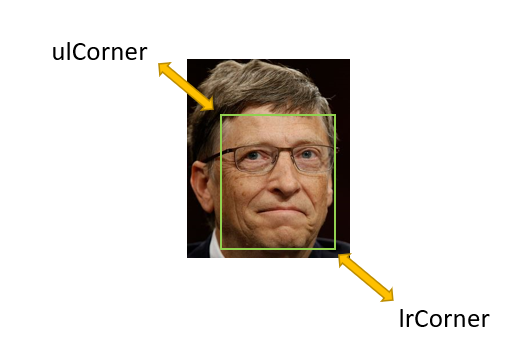


Figura 5 Ejemplo de esquinas del rectángulo (Fuente propia). Rostro de Bill Gates (Microsoft)

Sus atributos son

ulCorner: contiene las coordenadas X, Y de la esquina superior izquierda (upper left corner)

lrCorner: contiene las coordenadas X, Y de la esquina inferior derecha (lower right corner)

name: contendrá el nombre que la persona tras identificación

file: contiene el nombre del fichero que almacena el bitmap con la cara recortada, para poder enviar a Azure.

faceRecon

El módulo principal, se encargará de cargar la configuración, ejecutar el bucle principal y contener la función de identificación.

Atributos: se cargará desde un fichero INI la información para las siguientes variables:

* sKey: contiene la clave de suscripción de Azure
* baseURL: la URL base para la conexión con azure, el valor por defecto es 'https://northeurope.api.cognitive.microsoft.com/face/v1.0/'
* faceDetector: el archivo XML que contiene la información de filtros a emplear, se encuentran en la carpeta ‘utils’
* groupName: el nombre del grupo en Azure que contiene las personas cuyos datos ya han sido subidos. Por defecto será ‘conocidos’
* waitTime: tiempo de espera entre identificaciones

Función principal:

consiste en un bucle que se ejecuta siempre, salvo que el usuario ejecute Control+C o el sistema se apague. El bucle realiza las siguientes acciones como se observan en el siguiente pseudocodigo

Imagen // variable que almacena una imagen

Caras // array que almacena caras detectadas

while (true)

Imagen = capturar imagen

Caras.insertar (Imagen) //detecta, recortar

por cara en Caras

identificar cara

pintar cara en Imagen

Mostrar imagen

Función identifyFace(service, groupName, face)

Parámetros:

* service: el objeto que contiene las funciones para conectar con Azure
* groupName: nombre del grupo de identificación
* face: objeto de la clase Face

Esta función conecta con Azure, identifica la cara y pasa el nombre identificado a el objeto Face correspondiente. No devuelve ningún tipo de dato, solo modifica sobre la marcha cada objeto Face que este contenido en el array de caras.

Función paintImage(cv, font, img, face)

Parámetros:

* cv: objeto de OpenCV que contiene las funciones para gestión de imágenes
* font: fuente para escribir el texto sobre las imágenes
* img: imagen entera
* face: objeto de la clase Face

Esta función toma las coordenadas de un objeto Face y dibuja un rectángulo que enmarque la cara además de escribir el nombre correspondiente a la cara.

faceCrop

Este módulo contiene las funciones de gestión de las imágenes:

Funcion captureImage(camera)

Captura una imagen desde la cámara de Raspberry

Parametros:

* camera: objeto que contiene la cámara de la Raspberry

Retorno:

* img: un objeto PiRGBArray que contiene la información cruda de la imagen

Funcion faceDetect(faceDetector, image, cv2)

Detecta caras en una imagen

Parámetros:

* faceDetector: XML que contiene la información sobre el filtro Haar
* image: objeto PiRGBArray que contiene la imagen
* cv2: objeto con las funciones de OpenCV

Retorno:

* faceList: un array de objetos Face detectadas en la imagen

Funcion faceCrop(image, cv2, font, faceList)

Recorta caras de una imagen guardando cada una en un archivo, no devuelve ningún tipo de dato

Parámetros:

* image: objeto PiRGBArray que contiene la imagen
* cv2: objeto con las funciones de OpenCV
* font: fuente para escribir el texto sobre las imágenes
* faceList: un array de objetos Face detectadas en la imagen

FCTrain

Este modulo contiene varias funciones para el entrenamiento inicial del sistema, solo se describirá en detalle getNameByID, del resto solo se hará una descripción breve. Las otras funciones no se emplean durante el funcionamiento normal de FaceRecon pero son útiles para futuros usos y necesarias para los entrenamientos.

Función getNameByID(service, personGroup, personID)

Esta función cubre una necesidad no incluida en Azure, que es la relación entre el identificador asignado y el nombre real de la persona. Al identificar una cara Azure devuelve el ID del candidato, pero no su nombre, esta función devuelve el nombre real del candidato.

Parámetros:

* service: el objeto que contiene las funciones para conectar con Azure
* personGroup: nombre del grupo de identificación
* personID: id de una persona

Retorno:

* name: nombre real del candidato

Función testAzure (service)

Test de funcionamiento

Función getPersonID(service, personGroup, personName)

Obtiene el ID de una persona mediante el nombre real

Función createPersonGroup(service, personGroup)

Crea un grupo de personas

Función createPerson(service, personGroup, personName)

Crea una persona

Función addFace(service, personGroup, personName, image)

Añade una cara a una persona ya dada de alta

Función deletePerson(service, personGroup, name)

Elimina una persona y todas las caras asociadas

Función deletePersonGroup(service, personGroup)

Elimina un grupo de personas y toda información asociada

Función listPersons(service, personGroup)

Devuelve información sobre las personas de un grupo

Función detectPerson(service, personGroup, file)

Identifica una persona desde una imagen de archivo

* 1. FacePal

Una descripción genérica de la aplicación sería la siguiente:

1. Entrenamiento
   1. Comprobación con imágenes existentes

Para comprobar que el sistema esté funcionando y cumpla los objetivos de este proyecto, cabe plantearse las dos siguientes hipótesis

* al aumentar el número de imágenes, aumenta la confianza obtenida
* el sistema es capaz de distinguir personas con rasgos muy similares

Para ello se va a emplear el módulo de entrenamiento FRTrainer que forma parte de FaceRecon creada para el entrenamiento en la Raspberry, aunque se usará en Windows 10 por comodidad. Además, se diseñan cuatro pruebas que para poder confirmar ambas hipótesis:

* Bill Gates vs. Bill Gates
* Ben Linus vs. Ben Linus
* Bill Gates vs. Ben Linus
* Bill Gates vs. Bill Gates Impersonator

Todas estas pruebas se realizarán con imágenes

Prueba Bill Gates vs. Bill Gates

En esta primera prueba se emplearán 10 fotos de un personaje público del cual sea fácil obtener imágenes en Internet. El personaje elegido es Bill Gates cofundador de Microsoft, edad 62 años, sus rasgos son ojos azules, gafas, piel blanca, nariz mediana.



Fig. 10 Retratos Bill Gates (Fuente Google)

Para confirmar la primera hipótesis se irán realizando una serie de pruebas que pueden se explican con el siguiente pseudocodigo

crearPersona (“Bill Gates”)

// caraTest: imagen de prueba

// coleccionCaras: colección de fotos que el sistema desconoce

por cada Cara en coleccionCaras

añadir Cara a Persona

realizar entrenamiento

comparar con caraTest

imprimir candidato y confianza



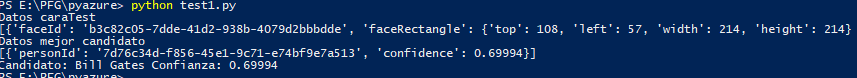
Figura 6 Imagen de Bill Gates empleada como caraTest (Fuente Microsoft)

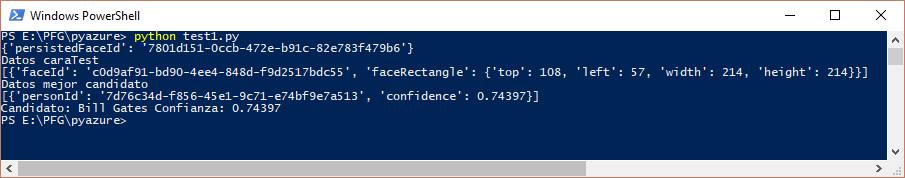
createPerson(CF,"person1", "Bill Gates")

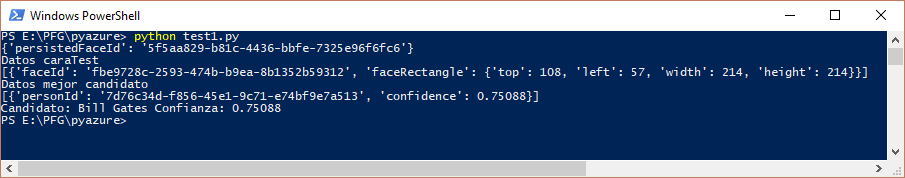
listPersons(CF, "person1")

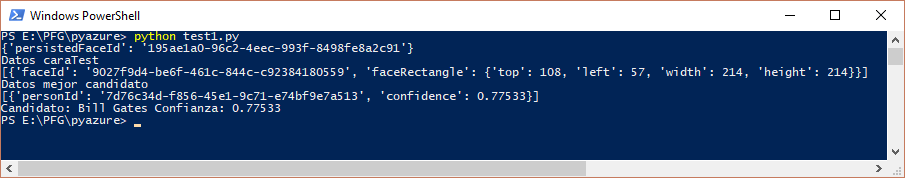


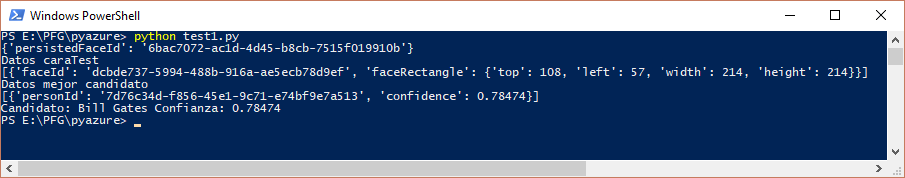


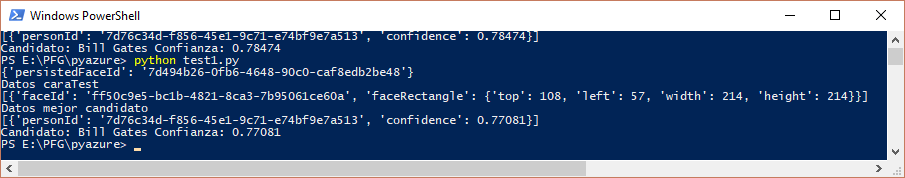


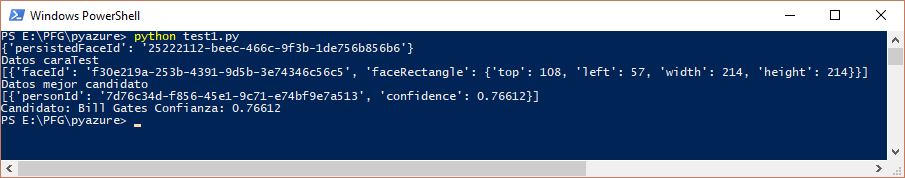


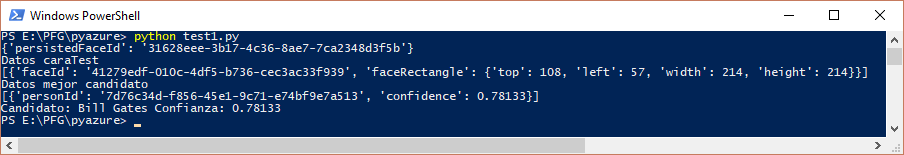


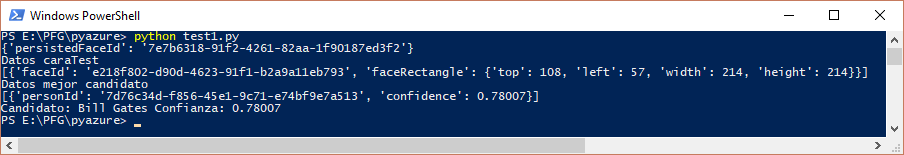


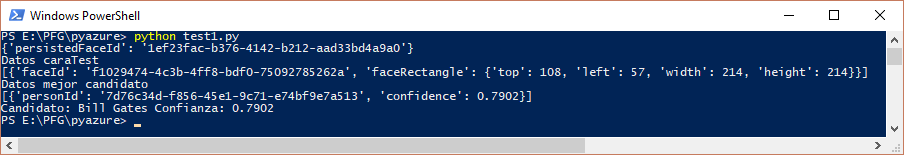












Como se puede comprobar, cuantas más imágenes se adjunten a la misma persona, mayor será la confianza obtenida, la primera hipótesis queda confirmada.

Aunque como se ve en el gráfico la tendencia es de crecimiento y tiende a estabilizarse, se pueden observar pequeñas fluctuaciones a las que conviene prestar atención, estas ocurren al añadir las imágenes bill5, bill6, y bill8, y pueden deberse a los siguientes motivos:

* bill5
  + el cristal de las gafas presenta demasiado brillo
  + la resolución de la imagen es baja comparada con las demás
* bill6
  + foto de juventud, aproximadamente 40 años, el resto de las fotos son más recientes
* bill8
  + la pose es distinta, la cabeza esta inclinada
  + la boca está más abierta que en el resto de la colección

Prueba Ben Linus vs. Ben Linus

Esta prueba está diseñada para poder verificar la primera hipótesis, es decir, es complementaria a la prueba anterior.

Para esta prueba, creamos a una nueva persona en el sistema que tiene una serie de rasgos similares a Bill Gates pero sin ser idéntico.

La persona elegida es Ben Linus principal villano de la serie “Lost” interpretado por el actor Michael Emerson, edad 63 años, ojos azules, gafas, nariz mediana y piel blanca.

Como en el caso anterior, se creará la persona de Ben Linus en el sistema y se le añadirán las 10 caras de la Figura 2.



Figura 7 Caras elegidas de Ben Linus (Fuente Google)

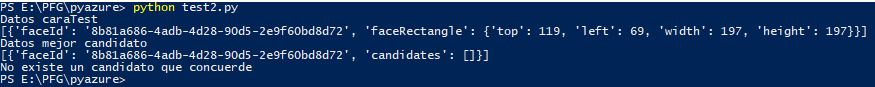
La diferencia en este caso es que las imágenes no serán tan homogéneas en tamaño ni forma como en la prueba anterior, de esta manera se podrá observar si el sistema es capaz de discernir el ruido que supone toda información fuera del área de la cara.

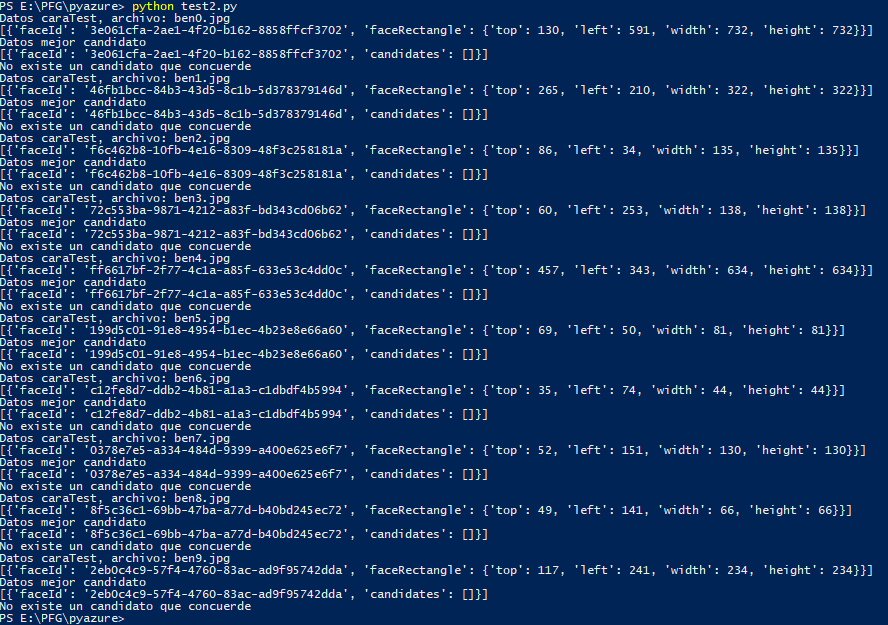
Adicionalmente se comprobara el caraTest que se observa en la Figura 3 y cada cara de Ben Linus individualmente antes incluso de crear la persona en el sistema para comprobar si alguna imagen tiene un parecido razonable con Bill Gates.



Figura 8 Ben Linus (Michael Emerson) (Fuente Wikipedia)

Tras la ejecución se obtienen los siguiente resultados:

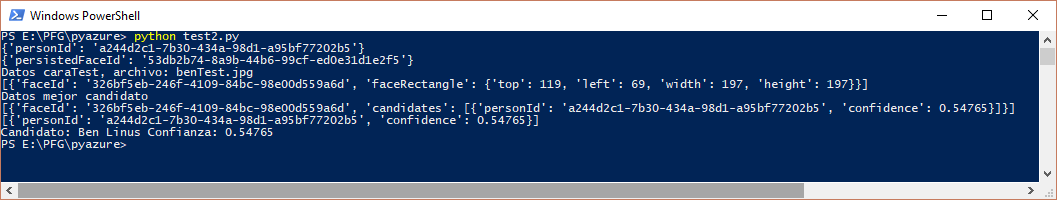


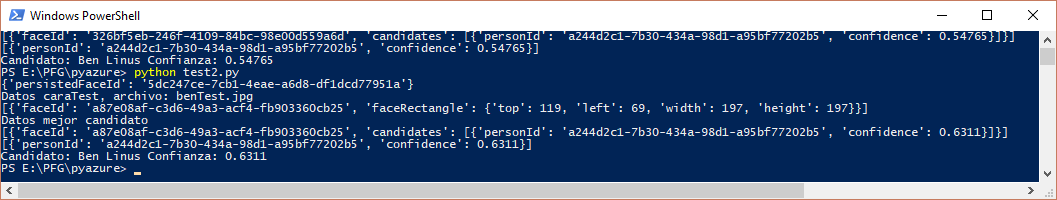


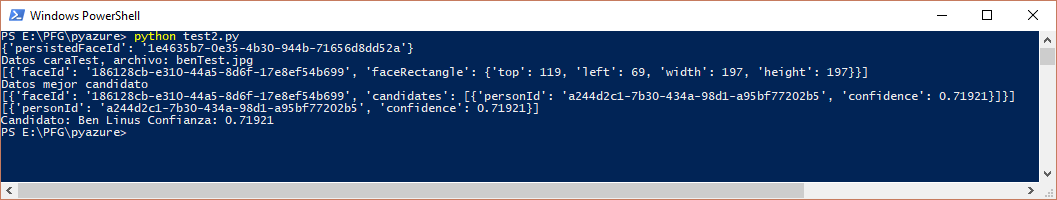
En ningún caso el sistema devuelve un candidato viable. Se debe tener en cuenta que Azure no devuelve a ningún candidato viable cuyo umbral de confianza este por debajo de 0.5.

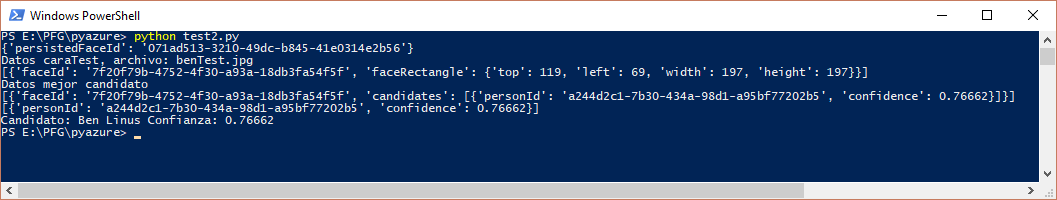
No significa que no exista algún tipo de similitud, sino que no es suficiente como para ser fiable. Por debajo de ese umbral (0.5) prácticamente sería como acertar la similitud tirando una moneda a cara o cruz.

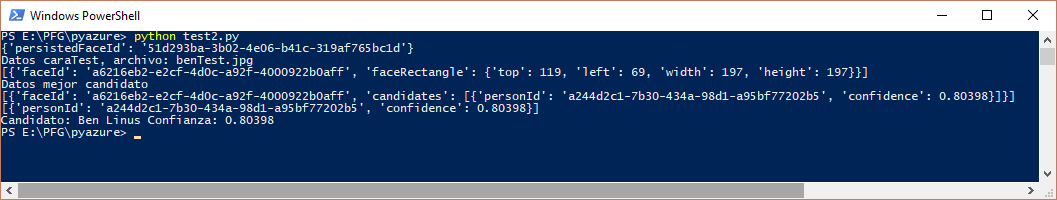
De forma similar a la prueba anterior########### se el test2

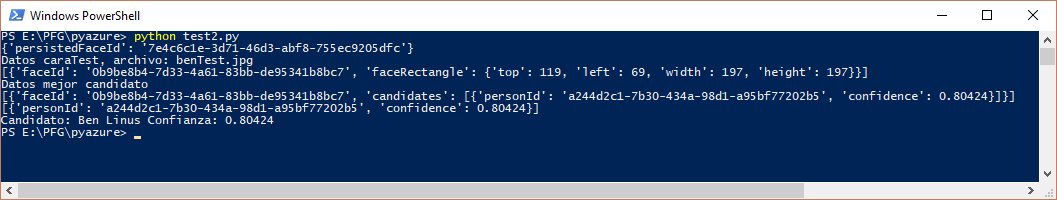


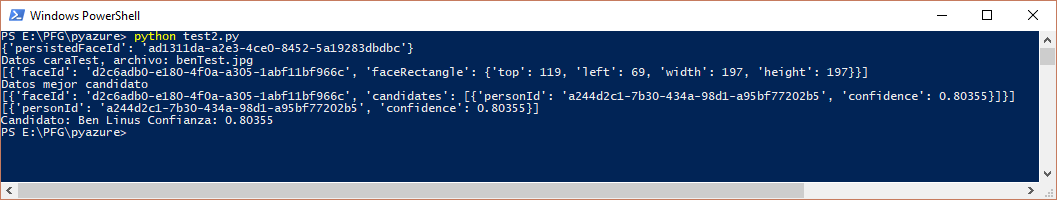


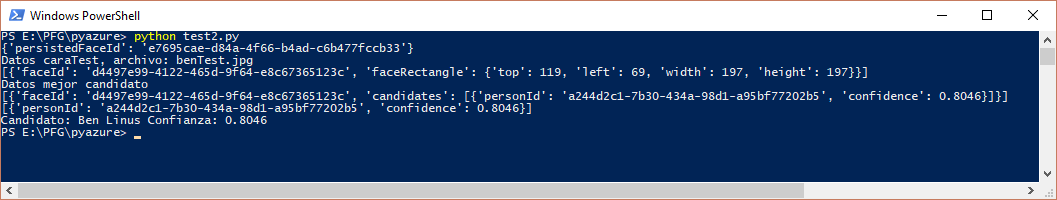


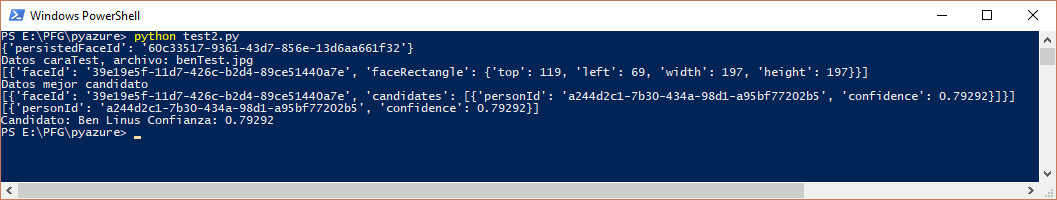


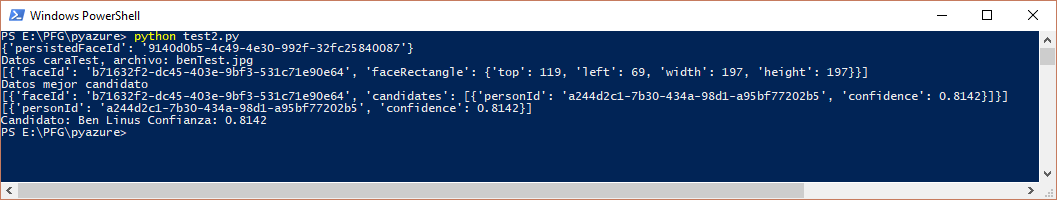












Como puede apreciarse en ##### el resultado es el esperado, a medida que aumenta el número de imágenes, aumenta la confianza que devuelve el sistema al usar el caraTest.

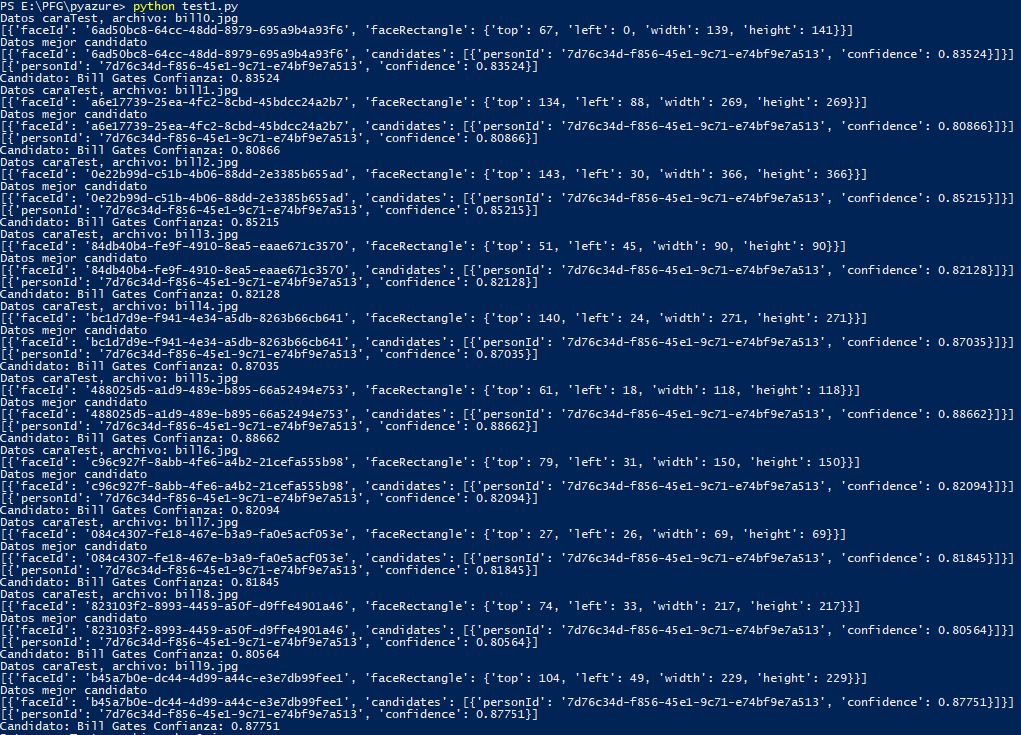
Un análisis más preciso del gráfico también revela otros datos de interés. El sistema no comienza a estabilizarse hasta la sexta imagen añadida, en la prueba Bill Gates vs. Bill Gates con la segunda imagen se conseguia un valor alto y casi estable de confianza.

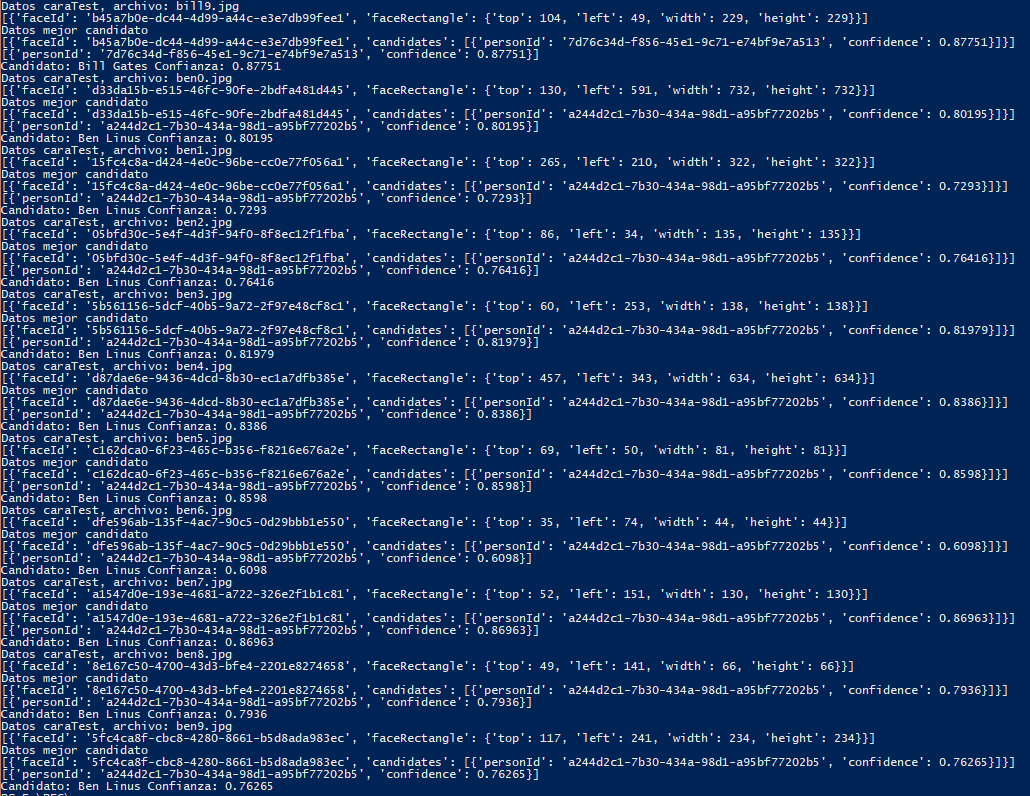
El motivo de este arranque tardío es debido a que la colección de imagen de Ben Linus es más heterogénea. La imagen ben0 (la primera que se sube al sistema) es oscura y Ben no tiene gafas, mientras que el caraTest es luminoso y con gafas, esto explica que el primer dato de confianza (0.54765) sea poco mejor que adivinar.

Pero incluso con una colección tan heterogenea (gafas, sin gafas, con sombrero, mucha luz, poca luz) el sistema al final del entrenamiento consigue una buena confianza (mayor del 80%) con el caraTest.

Prueba Bill Gates vs. Ben Linus

Para esta prueba se someterá al grupo que contiene a ambos personajes a todas las caras empleadas hasta el para el entrenamiento. De esta manera se podrá saber si alguna de las caras de Bill es reconocida como Ben y viceversa.





|  |  |  |
| --- | --- | --- |
| Archivos | **Bill Gates** | **Ben Linus** |
| **bill0** | 0.83524 | 0 |
| **bill1** | 0.80866 | 0 |
| **bill2** | 0.85215 | 0 |
| **bill3** | 0.82128 | 0 |
| **bill4** | 0.87035 | 0 |
| **bill5** | 0.88662 | 0 |
| **bill6** | 0.82094 | 0 |
| **bill7** | 0.81845 | 0 |
| **bill8** | 0.80564 | 0 |
| **bill9** | 0.87751 | 0 |
| **ben0** | 0 | 0.80195 |
| **ben1** | 0 | 0.72930 |
| **ben2** | 0 | 0.76416 |
| **ben3** | 0 | 0.81979 |
| **ben4** | 0 | 0.83860 |
| **ben5** | 0 | 0.85980 |
| **ben6** | 0 | 0.60980 |
| **ben7** | 0 | 0.86963 |
| **ben8** | 0 | 0.79360 |
| **ben9** | 0 | 0.76265 |

Tras la ejecución del test se puede observar que ninguna cara de Bill Gates es clasificada como Ben Linus, ni como segunda opción con peor confianza.

Del mismo modo ocurre que ninguna imagen de Ben Linus es identificada como Bill Gates.

Por tanto gracias a esta prueba podemos corroborar la segunda hipótesis.

Prueba Bill Gates vs. Bill Gates Impersonator

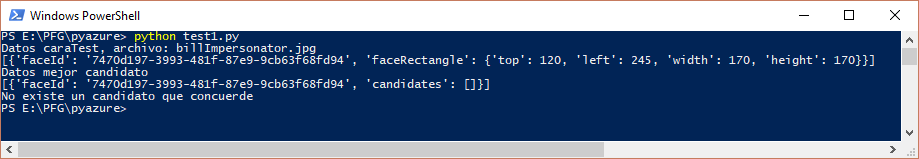
Por último, se decide hacer una prueba con una imagen curiosa encontrada en <http://www.lookalike.com>

Esta web ofrece el servicio de contratación de personas muy parecidas físicamente a personajes conocidos (actores, políticos, empresarios) y son entrenados para comportarse, vestirse y citar frases de dichos personajes.

En el catálogo, se ha encontrado a un imitador de Bill Gates bastante convincente, como se puede apreciar en la Figura 1. Como en los casos anteriores se somete la foto del imitador al método de verificación.



Figura 9 Imitador de Bill Gates (Fuente lookalike.com)



|  |  |  |
| --- | --- | --- |
| Archivos | **Bill Gates** | **Ben Linus** |
| **billImpersonator** | 0 | 0 |

El clasificador no encuentra a ningún candidato que tenga similitudes con el imitador, por tanto, la segunda hipótesis queda reforzada

1. Conclusiones
   1. Objetivos alcanzados
   2. Futuras ampliaciones
   3. Consideraciones

Una de las ideas durante la fase de análisis fue construir un servidor capaz de realizar el servicio de reconocimiento fácil de análisis fue construir un servidor capaz de realizar el servicio de reconocimiento facial, sin embargo, la complejidad de construir y montar todo el sistema queda fuera del alcance de este proyecto y es de una complejidad mayor de

Bibliografía

[] Camera Module - Raspberry Pi Documentation

<https://www.raspberrypi.org/documentation/hardware/camera/README.md>

[]

Listado de siglas, abreviaturas y acrónimos

API

SDK

JSON

LBP Local Binary Patterns

RAM

HDD

HDMI

WIFI

USB

open source

**GPIO** (*General Purpose Input/Output*

Gb

Ghz

GPU

GPS

IDE

Anexos

anexo??

Anexo materiales y métodos

Generacion de documentación -> Doxygen

Generacion de diagramas de flujo -> Dia

Lista de referencias y bibliografía consultada para la elaboración del proyecto. Todas las referencias deberán citarse dentro del cuerpo de la Memoria y sus Anexos. 

Listado de siglas, abreviaturas y acrónimos que se utilicen en el documento junto con sus respectivos significados. 

Anexos. Toda aquella información complementaria que tenga relación con el proyecto pero cuya naturaleza o forma no se ajuste a la estructura y contenidos del cuerpo del mismo.